Red Hat Directory Server Red Hat Directory Server 9

Deployment Guide

Updated for Directory Server 9.1.2

Last Updated: 2020-10-30

Red Hat Directory Server Red Hat Directory Server 9 Deployment Guide

Updated for Directory Server 9.1.2

Marc Muehlfeld
Red Hat Customer Content Services
mmuehlfeld@redhat.com

Petr Bokoč
Red Hat Customer Content Services

Ella Deon Lackey
Red Hat Customer Content Services

Legal Notice

Abstract

This guide is for planning a directory service. This documentation is no longer maintained. For details, see .

DEPRECATED DOCUMENTATION

IMPORTANT

Note that as of June 10, 2017, the support for Red Hat Directory Server 9 has ended. For details, see "Red Hat Directory Server Life Cycle policy". Red Hat recommends users of Directory Server 9 to update to the latest version.

Due to the end of the maintenance phase of this product, this documentation is no longer updated. Use it only as a reference!

PREFACE

The *Red Hat Directory Server Performance Tuning Guide* provides a solid foundation on the on concepts and configuration options for planning an effective directory service. The information provided here is intended for both designers and administrators.

1. DIRECTORY SERVER OVERVIEW

Red Hat Directory Server provides the following key features:

- Multi-master replication — Provides a highly available directory service for both read and write operations. Multi-master replication can be combined with simple and cascading replication scenarios to provide a highly flexible and scalable replication environment.
- Chaining and referrals — Increases the power of the directory by storing a complete logical view of the directory on a single server while maintaining data on a large number of Directory Servers transparently for clients.
- Roles and classes of service — Provides a flexible mechanism for grouping and sharing attributes between entries dynamically.
- Efficient access control mechanisms — Provides support for macros that dramatically reduce the number of access control statements used in the directory and increase the scalability of access control evaluation.
- Resource-limits by bind DN — Grants the power to control the amount of server resources allocated to search operations based on the bind DN of the client.
- Multiple databases — Provides a simple way of breaking down the directory data to simplify the implementation of replication and chaining in the directory service.
- Password policy and account lockout — Defines a set of rules that govern how passwords and user accounts are managed in the Directory Server.
- TLS and SSL — Provides secure authentication and communication over the network, using the Mozilla Network Security Services (NSS) libraries for cryptography.

The major components of Directory Server include the following:

- An LDAP server — The LDAP v3-compliant network daemon.
- Directory Server Console — A graphical management console that dramatically reduces the effort of setting up and maintaining the directory service.
- SNMP agent — Can monitor the Directory Server using the Simple Network Management Protocol (SNMP).

2. EXAMPLES AND FORMATTING

Each of the examples used in this guide, such as file locations and commands, have certain defined conventions.

2.1. Command and File Examples

All of the examples for Red Hat Directory Server commands, file locations, and other usage are given for Red Hat Enterprise Linux 5 (64-bit) systems. Be certain to use the appropriate commands and files for your platform.

Example 1. Example Command

To start the Red Hat Directory Server:

```
service dirsrv start
```

2.2. Brackets

Square brackets (**[]**) are used to indicate an alternative element in a name. For example, if a tool is available in **/usr/lib** on 32-bit systems and in **/usr/lib64** on 64-bit systems, then the tool location may be represented as **/usr/lib[64]**.

2.3. Client Tool Information

The tools for Red Hat Directory Server are located in the **/usr/bin** and the **/usr/sbin** directories.

IMPORTANT

The LDAP tools such as **ldapmodify** and **ldapsearch** from OpenLDAP use SASL connections by default. To perform a simple bind using a username and password, use the **-x** argument to disable SASL.

2.4. Text Formatting and Styles

Certain words are represented in different fonts, styles, and weights. Different character formatting is used to indicate the function or purpose of the phrase being highlighted.

Formatting Style	Purpose
Monospace font	Monospace is used for commands, package names, files and directory paths, and any text displayed in a prompt.
`Monospace with a background`	This type of formatting is used for anything entered or returned in a command prompt.
Italicized text	Any text which is italicized is a variable, such as *instance_name* or *hostname*. Occasionally, this is also used to emphasize a new term or other phrase.
Bolded text	Most phrases which are in bold are application names, such as **Cygwin**, or are fields or options in a user interface, such as a **User Name Here:** field or **Save** button.

Other formatting styles draw attention to important text.

NOTE

A note provides additional information that can help illustrate the behavior of the system or provide more detail for a specific issue.

IMPORTANT

Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.

WARNING

A warning indicates potential data loss, as may happen when tuning hardware for maximum performance.

3. ADDITIONAL READING

The *Red Hat Directory Server Deployment Guide* describes many of the basic directory and architectural concepts that you need to deploy, install, and administer a directory service successfully.

When you are familiar with Directory Server concepts and have done some preliminary planning for your directory service, install the Directory Server. The instructions for installing the various Directory Server components are contained in the *Red Hat Directory Server Installation Guide* . Many of the scripts and commands used to install and administer the Directory Server are explained in detail in the *Red Hat Directory Server Configuration, Command, and File Reference*.

The *Directory Server Administrator's Guide* describes how to set up, configure, and administer Red Hat Directory Server and its contents.

The document set for Directory Server contains the following guides:

- *Red Hat Directory Server Release Notes* contain important information on new features, fixed bugs, known issues and workarounds, and other important deployment information for this specific version of Directory Server.
- *Red Hat Directory Server Deployment Guide* provides an overview for planning a deployment of the Directory Server.
- *Red Hat Directory Server Administrator's Guide* contains procedures for the day-to-day maintenance of the directory service. Includes information on configuring server-side plug-ins.
- *Red Hat Directory Server Configuration, Command, and File Reference* provides reference information on the command-line scripts, configuration attributes, schema elements, and log files shipped with Directory Server.
- *Red Hat Directory Server Installation Guide* contains procedures for installing your Directory Server as well as procedures for migrating from a previous installation of Directory Server.
- *Red Hat Directory Server Plug-in Guide* describes how to write server plug-ins in order to customize and extend the capabilities of Directory Server.
- The *Red Hat Directory Server Performance Tuning Guide* contains features to monitor overall Directory Server and database performance, to tune attributes for specific operations, and to tune the server and database for optimum performance.

For the latest information about Directory Server, including current release notes, complete product documentation, technical notes, and deployment information, see the Red Hat Directory Server documentation site at https://access.redhat.com/site/documentation/Red_Hat_Directory_Server/.

4. GIVING FEEDBACK

If there is any error in this *Deployment Guide* or there is any way to improve the documentation, please let us know. Bugs can be filed against the documentation for Red Hat Directory Server through Bugzilla, http://bugzilla.redhat.com/bugzilla. Make the bug report as specific as possible, so we can be more effective in correcting any issues:

1. Select the Red Hat Directory Server product.
2. Set the component to **Doc - deployment-guide**.
3. Set the version number to 9.0.
4. For errors, give the page number (for the PDF) or URL (for the HTML), and give a succinct description of the problem, such as incorrect procedure or typo.

   For enhancements, put in what information needs to be added and why.

5. Give a clear title for the bug. For example, **"Incorrect command example for setup script options"** is better than **"Bad example"**.

We appreciate receiving any feedback — requests for new sections, corrections, improvements, enhancements, even new ways of delivering the documentation or new styles of docs. You are welcome to contact Red Hat Content Services directly at docs@redhat.com.

CHAPTER 1. INTRODUCTION TO DIRECTORY SERVICES

Red Hat Directory Server provides a centralized directory service for an intranet, network, and extranet information. Directory Server integrates with existing systems and acts as a centralized repository for the consolidation of employee, customer, supplier, and partner information. Directory Server can even be extended to manage user profiles, preferences, and authentication.

This chapter describes the basic ideas and concepts for understanding what a directory service does to help begin designing the directory service.

1.1. ABOUT DIRECTORY SERVICES

The term *directory service* refers to the collection of software, hardware, and processes that store information about an enterprise, subscribers, or both, and make that information available to users. A directory service consists of at least one instance of Directory Server and at least one directory client program. Client programs can access names, phone numbers, addresses, and other data stored in the directory service.

An example of a directory service is a domain name system (DNS) server. A DNS server maps computer hostnames to IP addresses. Thus, all of the computing resources (hosts) become clients of the DNS server. Mapping hostnames allows users of computing resources to easily locate computers on a network by remembering hostnames rather than IP addresses. A limitation of a DNS server is that it stores only two types of information: names and IP addresses. A true directory service stores virtually unlimited types of information.

Directory Server stores all user and network information in a single, network-accessible repository. Many kinds of different information can be stored in the Directory Server:

- Physical device information, such as data about the printers in an organization, such as location, color or black and white, manufacturer, date of purchase, and serial number.
- Public employee information, such as name, email address, and department.
- Private employee information, such as salary, government identification numbers, home addresses, phone numbers, and pay grade.
- Contract or account information, such as the name of a client, final delivery date, bidding information, contract numbers, and project dates.

Directory Server serves the needs of a wide variety of applications. It also provides a standard protocol and application programming interfaces (APIs) to access the information it contains.

1.1.1. About Global Directory Services

Directory Server provides global directory services, which means that it provides information to a wide variety of applications. Rather than attempting to unify proprietary databases bundled with different applications, which is an administrative burden, Directory Server is a single solution to manage the same information.

For example, a company is running three different proprietary email systems, each with its own proprietary directory service. If users change their passwords in one directory, the changes are not automatically replicated in the others. Managing multiple instances of the same information results in increased hardware and personnel costs; the increased maintenance overhead is referred to as the *n+1 directory problem*.

A global directory service solves the n+1 directory problem by providing a single, centralized repository of directory information that any application can access. However, giving a wide variety of applications access to the directory service requires a network-based means of communicating between the applications and the directory service. Directory Server uses LDAP for applications to access to its global directory service.

1.1.2. About LDAP

LDAP provides a common language that client applications and servers use to communicate with one another. LDAP is a "lightweight" version of the Directory Access Protocol (DAP) described by the ISO X.500 standard. DAP gives any application access to the directory through an extensible and robust information framework but at a high administrative cost. DAP uses a communications layer that is not the Internet standard protocol and has complex directory-naming conventions.

LDAP preserves the best features of DAP while reducing administrative costs. LDAP uses an open directory access protocol running over TCP/IP and simplified encoding methods. It retains the data model and can support millions of entries for a modest investment in hardware and network infrastructure.

1.2. INTRODUCTION TO DIRECTORY SERVER

Red Hat Directory Server is comprised of several components. The core of the directory itself is the server that

implements the LDAP protocol. Red Hat Directory Server has a client-side graphical user interface on top of the LDAP server that allows end-users to search and change entries in the directory. Other LDAP clients, both third-party programs and custom programs written using the Mozilla LDAP SDK and the OpenLDAP SDK, can be used with Red Hat Directory Server or to integrate other applications with Red Hat Directory Server.

When the Red Hat Directory Server is installed, it has these elements:

- The core Directory Server LDAP server, the LDAP v3-compliant network daemon (**ns-slapd**) and all of the associated plug-ins, command-line tools for managing the server and its databases, and its configuration and schema files. For more information about the command-line tools, see the *Directory Server Configuration, Command, and File Reference*.
- Admin Server, a web server which controls the different portals that access the LDAP server. For more information about the Admin Server, see *Administrator's Guide*.
- Directory Server Console, a graphical management console that dramatically reduces the effort of setting up and maintaining the directory service. For more information about the Directory Server Console, see *Administrator's Guide*.
- SNMP agent to monitor the Directory Server using the Simple Network Management Protocol (SNMP). For more information about SNMP monitoring, see the *Directory Server Administrator's Guide* .

Without adding other LDAP client programs, Directory Server can provide the foundation for an intranet or extranet. Compatible server applications use the directory as a central repository for shared server information, such as employee, customer, supplier, and partner data.

Directory Server can manage user authentication, create access control, set up user preferences, and centralize user management. In hosted environments, partners, customers, and suppliers can manage their own portions of the directory, reducing administrative costs.

1.2.1. Overview of the Server Frontend

Directory Server is a multi-threaded application. This means that multiple clients can bind to the server at the same time over the same network. As directory services grow to include larger numbers of entries or geographically-dispersed clients, they also include multiple Directory Servers placed in strategic places around the network.

The server frontend of Directory Server manages communications with directory client programs. Multiple client programs can communicate with the server using both LDAP over TCP/IP (Internet traffic protocols) and LDAP over Unix sockets (LDAPI). The Directory Server can establish a *secure* (encrypted) connection with SSL/TLS, depending on whether the client negotiates the use of Transport Layer Security (TLS) for the connection.

When communication takes place with TLS, the communication is usually encrypted. If clients have been issued certificates, TLS/SSL can be used by Directory Server to confirm that the client has the right to access the server. TLS/SSL is used to perform other security activities, such as message integrity checks, digital signatures, and mutual authentication between servers.

NOTE

Directory Server runs as a daemon; the process is **ns-slapd**.

1.2.2. Server Plug-ins Overview

Directory Server relies on *plug-ins* to add functionality to the core server. For example, a database layer is a plug-in. Directory Server has plug-ins for replication, chaining databases, and other different directory functions.

Generally, a plug-in can be disabled, particularly plug-ins the extend the server functionality. When disabled, the plug-in's configuration information remains in the directory, but its function is not used by the server. Depending on what the directory is supposed to do, any of the plug-ins provided with Directory Server can be enabled to extend the Directory Server functionality. (Plug-ins related to the core directory service operations, like back end database plug-in, naturally cannot be disabled.)

For more information on the default plug-ins with Directory Server and the functions available for writing custom plug-ins, see the *Directory Server Plug-in Guide* .

1.2.3. Overview of the Basic Directory Tree

The directory tree, also known as a directory information tree (DIT), mirrors the tree model used by most file systems, with the tree's root, or first entry, appearing at the top of the hierarchy. During installation, Directory Server creates a default directory tree.

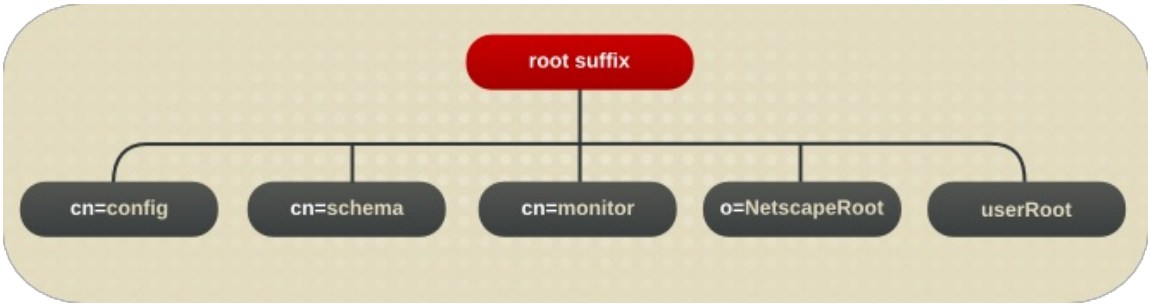


Figure 1.1. Layout of Default Directory Server Directory Tree

The root of the tree is called the *root suffix*. For information about naming the root suffix, see Section 4.2.1, "Choosing a Suffix".

After a standard installation, the directory contains three subtrees under the root suffix:

- **cn=config**, the subtree containing information about the server's internal configuration.
- **o=NetscapeRoot**, the subtree containing the configuration information of the Directory Server and Admin Server.

  NOTE

  When additional instances of Directory Server are installed, they can be configured not to have an **o=NetscapeRoot** database; in that case, the instances use a *configuration directory* (or the **o=NetscapeRoot** subtree) on another server. See the *Directory Server Installation Guide* for more information about choosing the location of the configuration directory.

- **cn=monitor**, the subtree containing Directory Server server and database monitoring statistics.
- **cn=schema**, the subtree containing the schema elements currently loaded in the server.
- *user_suffix*, the suffix for the default user database created when the Directory Server is setup. The name of the suffix is defined by the user when the server is created; the name of the associated database is **userRoot**. The database can be populated with entries by importing an LDIF file at setup or entries can be added to it later.

  The *user_suffix* suffix frequently has a ***dc*** naming convention, like **dc=example,dc=com**. Another common naming attribute is the ***o*** attribute, which is used for an entire organization, like **o=example.com**.

The default directory tree can be extended to add any data relevant to the directory installation. For more information about directory trees, see Chapter 4, *Designing the Directory Tree*.

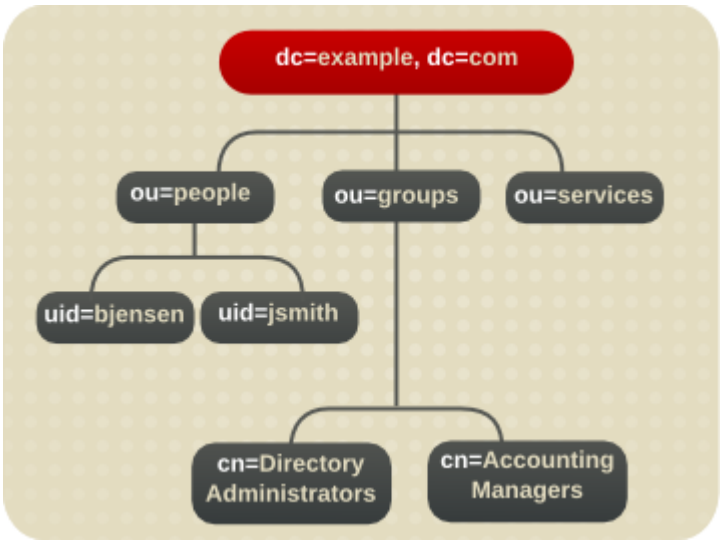


Figure 1.2. Expanded Directory Tree for Example Corp.

1.3. DIRECTORY SERVER DATA STORAGE

The *database* is the basic unit of storage, performance, replication, and indexing. All Directory Server operations –

importing, exporting, backing up, restoring, and indexing entries — are performed on the database. Directory data are stored in an LDBM database. The LDBM database is implemented as a plug-in that is automatically installed with the directory and is enabled by default.

By default, Directory Server uses one back end database instance for a root suffix, and, by default, there are two databases, **o=NetscapeRoot** for configuration entries and **userRoot** for directory entries. A single database is sufficient to contain the directory tree. This database can manage millions of entries.

This database supports advanced methods of backing up and restoring data, in order to minimize risk to data.

Multiple databases can be used to support the whole Directory Server deployment. Information is distributed across the databases, allowing the server to hold more data than can be stored in a single database.

1.3.1. About Directory Entries

LDAP Data Interchange Format (LDIF) is a standard text-based format for describing directory entries. An *entry* consists of a number of lines in the LDIF file (also called a *stanza*), which contains information about an object, such as a person in the organization or a printer on the network.

Information about the entry is represented in the LDIF file by a set of attributes and their values. Each entry has an object class attribute that specifies the kind of object the entry describes and defines the set of additional attributes it contains. Each attribute describes a particular trait of an entry.

For example, an entry might be of object class **organizationalPerson**, indicating that the entry represents a person within an organization. This object class supports the ***givenname*** and ***telephoneNumber*** attributes. The values assigned to these attributes give the name and phone number of the person represented by the entry.

Directory Server also uses read-only attributes that are calculated by the server. These attributes are called *operational attributes*. The administrator can manually set operational attributes that can be used for access control and other server functions.

1.3.1.1. Performing Queries on Directory Entries

Entries are stored in a hierarchical structure in the directory tree. LDAP supports tools that query the database for an entry and request all entries below it in the directory tree. The root of this subtree is called the *base distinguished name*, or base DN. For example, if performing an LDAP search request specifying a base DN of **ou=people,dc=example,dc=com**, then the search operation examines only the **ou=people** subtree in the **dc=example,dc=com** directory tree.

Not all entries are automatically returned in response to an LDAP search, however, because administrative entries (which have the **ldapsubentry** object class) are not returned by default with LDAP searches. Administrative object, for example, can be entries used to define a role or a class of service. To include these entries in the search response, clients need to search specifically for entries with the **ldapsubentry** object class. See Section 4.3.2, “About Roles” for more information about roles and Section 5.3, “About Classes of Service” for more information about class of service.

1.3.2. Distributing Directory Data

When various parts of the directory tree are stored in separate databases, the directory can process client requests in parallel, which improves performance. The databases can even be located on different machines to further improve performance.

Distributed data are connected by a special entry in a subtree of the directory, called a *database link*, which point to data stored remotely. When a client application requests data from a database link, the database link retrieves the data from the remote database and returns it to the client. All LDAP operations attempted below this entry are sent to the remote machine. This method is called *chaining*.

Chaining is implemented in the server as a plug-in, which is enabled by default.

1.4. DIRECTORY DESIGN OVERVIEW

Planning the directory service before actual deployment is the most important task to ensure the success of the directory. The design process involves gathering data about the directory requirements, such as environment and data sources, users, and the applications that use the directory. This information is integral to designing an effective directory service because it helps identify the arrangement and functionality required.

The flexibility of Directory Server means the directory design can be reworked to meet unexpected or changing requirements, even after the Directory Server is deployed.

1.4.1. Design Process Outline

1. Chapter 2, *Planning the Directory Data*

   The directory contains data such as user names, telephone numbers, and group details. This chapter analyzes the various sources of data in the organization and understand their relationship with one another. It describes the types of data that can be stored in the directory and other tasks to perform to design the contents of the Directory Server.

2. Chapter 3, *Designing the Directory Schema*

   The directory is designed to support one or more directory-enabled applications. These applications have requirements of the data stored in the directory, such as the file format. The directory schema determines the characteristics of the data stored in the directory. The standard schema shipped with Directory Server is introduced in this chapter, as well as a description of how to customize the schema and tips for maintaining a consistent schema.

3. Chapter 4, *Designing the Directory Tree*

   Along with determining *what* information is contained in the Directory Server, it is important to determine *how* that information is going to be organized and referenced. This chapter introduces the directory tree and gives an overview of the design of the data hierarchy. Sample directory tree designs are also provided.

4. Chapter 6, *Designing the Directory Topology*

   Topology design means how the directory tree is divided among multiple physical Directory Servers and how these servers communicate with one another. The general principles behind design, using multiple databases, the mechanisms available for linking the distributed data together, and how the directory itself keeps track of distributed data are all described in this chapter.

5. Chapter 7, *Designing the Replication Process*

   When replication is used, multiple Directory Servers maintain the same directory data to increase performance and provide fault tolerance. This chapter describes how replication works, what kinds of data can be replicated, common replication scenarios, and tips for building a high-availability directory service.

6. Chapter 8, *Designing Synchronization*

   The information stored in the Red Hat Directory Server can by synchronized with information stored in Microsoft Active Directory databases for better integration with a mixed-platform infrastructure. This chapter describes how synchronization works, what kinds of data can be synched, and considerations for the type of information and locations in the directory tree which are best for synchronization.

7. Chapter 9, *Designing a Secure Directory*

   Finally, plan how to protect the data in the directory and design the other aspects of the service to meet the security requirements of the users and applications. This chapter covers common security threats, an overview of security methods, the steps involved in analyzing security needs, and tips for designing access controls and protecting the integrity of the directory data.

1.4.2. Deploying the Directory

The first step to deploying the Directory Server is installing a test server instance to make sure the service can handle the user load. If the service is not adequate in the initial configuration, adjust the design and test it again. Adjust the design until it is a robust service that you can confidently introduce to the enterprise.

For a comprehensive overview of creating and implementing a directory pilot, see *Understanding and Deploying LDAP Directory Services* (T. Howes, M. Smith, G. Good, Macmillan Technical Publishing, 1999).

After creating and tuning a successful test Directory Server instance, develop a plan to move the directory service to production which covers the following considerations:

- An estimate of the required resources
- A schedule of what needs to be accomplished and when
- A set of criteria for measuring the success of the deployment

See the *Directory Server Installation Guide* for information on installing the directory service and the *Directory Server Administrator's Guide* for information on administering and maintaining the directory.

1.5. OTHER GENERAL DIRECTORY RESOURCES

The following publications have very detailed and useful information about directories, LDAP, and LDIF:

- RFC 2849: The LDAP Data Interchange Format (LDIF) Technical Specification, http://www.ietf.org/rfc/rfc2849.txt
- RFC 2251: Lightweight Directory Access Protocol (v3), http://www.ietf.org/rfc/rfc2251.txt
- *Understanding and Deploying LDAP Directory Services* . T. Howes, M. Smith, G. Good, Macmillan Technical Publishing, 1999.

All of the Red Hat Directory Server documentation, available at http://redhat.com/docs/manuals/dir-server, also contain high-level concepts about using LDAP and managing directory services, as well as Directory Server-specific information.

CHAPTER 2. PLANNING THE DIRECTORY DATA

The data stored in the directory may include user names, email addresses, telephone numbers, and information about groups users are in, or it may contain other types of information. The type of data in the directory determines how the directory is structured, who is given access to the data, and how this access is requested and granted.

This chapter describes the issues and strategies behind planning the directory's data.

2.1. INTRODUCTION TO DIRECTORY DATA

Some types of data are better suited to the directory than others. Ideal data for a directory has some of the following characteristics:

- It is read more often than written.
- It is expressible in attribute-data format (for example, **surname=jensen**).
- It is of interest to more than one person or group. For example, an employee's name or the physical location of a printer can be of interest to many people and applications.
- It will be accessed from more than one physical location.

For example, an employee's preference settings for a software application may not seem to be appropriate for the directory because only a single instance of the application needs access to the information. However, if the application is capable of reading preferences from the directory and users might want to interact with the application according to their preferences from different sites, then it is very useful to include the preference information in the directory.

2.1.1. Information to Include in the Directory

Any descriptive or useful information about a person or asset can be added to an entry as an attribute. For example:

- Contact information, such as telephone numbers, physical addresses, and email addresses.
- Descriptive information, such as an employee number, job title, manager or administrator identification, and job-related interests.
- Organization contact information, such as a telephone number, physical address, administrator identification, and business description.
- Device information, such as a printer's physical location, type of printer, and the number of pages per minute that the printer can produce.
- Contact and billing information for a corporation's trading partners, clients, and customers.
- Contract information, such as the customer's name, due dates, job description, and pricing information.
- Individual software preferences or software configuration information.
- Resource sites, such as pointers to web servers or the file system of a certain file or application.

Using the Directory Server for more than just server administration requires planning what other types of information to store in the directory. For example:

- Contract or client account details
- Payroll data
- Physical device information
- Home contact information
- Office contact information for the various sites within the enterprise

2.1.2. Information to Exclude from the Directory

Red Hat Directory Server is excellent for managing large quantities of data that client applications read and write, but it is not designed to handle large, unstructured objects, such as images or other media. These objects should be maintained in a file system. However, the directory can store pointers to these kinds of applications by using pointer URLs to FTP, HTTP, and other sites.

2.2. DEFINING DIRECTORY NEEDS

When designing the directory data, think not only of the data that is currently required but also how the directory (and organization) is going to change over time. Considering the future needs of the directory during the design process influences how the data in the directory are structured and distributed.

Look at these points:

- What should be put in the directory today?
- What immediate problem is solved by deploying a directory?
- What are the immediate needs of the directory-enabled application being used?
- What information is going to be added to the directory in the near future? For example, an enterprise might use an accounting package that does not currently support LDAP but will be LDAP-enabled in a few months. Identify the data used by LDAP-compatible applications, and plan for the migration of the data into the directory as the technology becomes available.
- What information might be stored in the directory in the future? For example, a hosting company may have future customers with different data requirements than their current customers, such as needing to store images or media files. While this is the hardest answer to anticipate, doing so may pay off in unexpected ways. At a minimum, this kind of planning helps identify data sources that might not otherwise have been considered.

2.3. PERFORMING A SITE SURVEY

A site survey is a formal method for discovering and characterizing the contents of the directory. Budget plenty of time for performing a site survey, as preparation is the key to the directory architecture. The site survey consists of a number of tasks:

- Identify the applications that use the directory.

  Determine the directory-enabled applications deployed across the enterprise and their data needs.

- Identify data sources.

  Survey the enterprise and identify sources of data, such as Active Directory, other LDAP servers, PBX systems, human resources databases, and email systems.

- Characterize the data the directory needs to contain.

  Determine what objects should be present in the directory (for example, people or groups) and what attributes of these objects to maintain in the directory (such as usernames and passwords).

- Determine the level of service to provide.

  Decide how available the directory data needs to be to client applications, and design the architecture accordingly. How available the directory needs to be affects how data are replicated and how chaining policies are configured to connect data stored on remote servers.

  See Chapter 7, *Designing the Replication Process* for more information about replication and Section 6.1, "Topology Overview" for more information on chaining.

- Identify a data master.

  A data master contains the primary source for directory data. This data might be mirrored to other servers for load balancing and recovery purposes. For each piece of data, determine its data master.

- Determine data ownership.

  For each piece of data, determine the person responsible for ensuring that the data is up-to-date.

- Determine data access.

  If data are imported from other sources, develop a strategy for both bulk imports and incremental updates. As a part of this strategy, try to master data in a single place, and limit the number of applications that can change the data. Also, limit the number of people who write to any given piece of data. A smaller group ensures data integrity while reducing the administrative overhead.

- Document the site survey.

Because of the number of organizations that can be affected by the directory, it may be helpful to create a directory deployment team that includes representatives from each affected organization to perform the site survey.

Corporations generally have a human resources department, an accounting or accounts receivable department, manufacturing organizations, sales organizations, and development organizations. Including representatives from each of these organizations can help the survey process. Furthermore, directly involving all the affected organizations can help build acceptance for the migration from local data stores to a centralized directory.

2.3.1. Identifying the Applications That Use the Directory

Generally, the applications that access the directory and the data needs of these applications drive the planning of the directory contents. Many common applications use the directory:

- *Directory browser applications, such as online telephone books* . Decide what information (such as email addresses, telephone numbers, and employee name) users need, and include it in the directory.
- *Email applications, especially email servers* . All email servers require email addresses, user names, and some routing information to be available in the directory. Others, however, require more advanced information such as the place on disk where a user's mailbox is stored, vacation notification information, and protocol information (IMAP versus POP, for example).
- *Directory-enabled human resources applications* . These require more personal information such as government identification numbers, home addresses, home telephone numbers, birth dates, salary, and job title.
- *Microsoft Active Directory*. Through Windows User Sync, Windows directory services can be integrated to function in tandem with the Directory Server. Both directories can store user information (user names and passwords, email addresses, telephone numbers) and group information (members). Style the Directory Server deployment after the existing Windows server deployment (or vice versa) so that the users, groups, and other directory data can be smoothly synchronized.

When examining the applications that will use the directory, look at the types of information each application uses. The following table gives an example of applications and the information used by each:

Table 2.1. Example Application Data Needs

Application	Class of Data	Data
Phonebook	People	Name, email address, phone number, user ID, password, department number, manager, mail stop.
Web server	People, groups	User ID, password, group name, groups members, group owner.
Calendar server	People, meeting rooms	Name, user ID, cube number, conference room name.

After identifying the applications and information used by each application, it is apparent that some types of data are used by more than one application. Performing this kind of exercise during the data planning stage can help to avoid data redundancy problems in the directory, and show more clearly what data directory-dependent applications require.

The final decision about the types of data maintained in the directory and when the information is migrated to the directory is affected by these factors:

- The data required by various legacy applications and users
- The ability of legacy applications to communicate with an LDAP directory

2.3.2. Identifying Data Sources

To identify all of the data to include in the directory, perform a survey of the existing data stores. The survey should include the following:

- Identify organizations that provide information.

  Locate all the organizations that manage information essential to the enterprise. Typically, this includes the information services, human resources, payroll, and accounting departments.

- Identify the tools and processes that are information sources.

  Some common sources for information are networking operating systems (Windows, Novell Netware, UNIX NIS), email systems, security systems, PBX (telephone switching) systems, and human resources applications.

- Determine how centralizing each piece of data affects the management of data.

  Centralized data management can require new tools and new processes. Sometimes centralization requires increasing staff in some organizations while decreasing staff in others.

During the survey, consider developing a matrix that identifies all of the information sources in the enterprise, similar to Table 2.2, " Example Information Sources":

Table 2.2. Example Information Sources

Data Source	Class of Data	Data
Human resources database	People	Name, address, phone number, department number, manager.
Email system	People, Groups	Name, email address, user ID, password, email preferences.
Facilities system	Facilities	Building names, floor names, cube numbers, access codes.

2.3.3. Characterizing the Directory Data

All of the data identified to include in the directory can be characterized according to the following general points:

- Format
- Size
- Number of occurrences in various applications
- Data owner
- Relationship to other directory data

Study each kind of data to include in the directory to determine what characteristics it shares with the other pieces of data. This helps save time during the schema design stage, described in more detail in Chapter 3, *Designing the Directory Schema*.

A good idea is to use a table, similar to Table 2.3, "Directory Data Characteristics", which characterizes the directory data.

Table 2.3. Directory Data Characteristics

Data	Format	Size	Owner	Related to
Employee Name	Text string	128 characters	Human resources	User's entry
Fax number	Phone number	14 digits	Facilities	User's entry
Email address	Text	Many character	IS department	User's entry

2.3.4. Determining Level of Service

The level of service provided depends on the expectations of the people who rely on directory-enabled applications. To determine the level of service each application expects, first determine how and when the application is used.

As the directory evolves, it may need to support a wide variety of service levels, from production to mission critical. It can be difficult raising the level of service after the directory is deployed, so make sure the initial design can meet the future needs.

For example, if the risk of total failure must be eliminated, use a multi-master configuration, where several suppliers exist for the same data.

2.3.5. Considering a Data Master

A *data master* is a server that is the master source of data. Any time the same information is stored in multiple locations, the data integrity can be degraded. A data master makes sure all information stored in multiple locations is consistent and accurate. There are several scenarios that require a data master:

- Replication among Directory Servers
- Synchronization between Directory Server and Active Directory
- Independent client applications which access the Directory Server data

Consider the master source of the data if there are applications that communicate indirectly with the directory. Keep the processes for changing data, and the places from which the data can be changed, as simple as possible. After deciding on a single site to master a piece of data, use the same site to master all of the other data contained there. A single site simplifies troubleshooting if the databases lose synchronization across the enterprise.

There are different ways to implement data mastering:

- Master the data in both the directory and all applications that do not use the directory.

  Maintaining multiple data masters does not require custom scripts for moving data in and out of the directory and the other applications. However, if data changes in one place, someone has to change it on all the other sites. Maintaining master data in the directory and all applications not using the directory can result in data being unsynchronized across the enterprise (which is what the directory is supposed to prevent).

- Master the data in some application other than the directory, and then write scripts, programs, or gateways to import that data into the directory.

  Mastering data in non-directory applications makes the most sense if there are one or two applications that are already used to master data, and the directory will be used only for lookups (for example, for online corporate telephone books).

How master copies of the data are maintained depends on the specific directory needs. However, regardless of how data masters are maintained, keep it simple and consistent. For example, do not attempt to master data in multiple sites, then automatically exchange data between competing applications. Doing so leads to a "last change wins" scenario and increases the administrative overhead.

For example, the directory is going to manage an employee's home telephone number. Both the LDAP directory and a human resources database store this information. The human resources application is LDAP-enabled, so an application can be written that automatically transfers data from the LDAP directory to the human resources database, and vice versa.

Attempting to master changes to that employee's telephone number in both the LDAP directory and the human resources data, however, means that the last place where the telephone number was changed overwrites the information in the other database. This is only acceptable as long as the last application to write the data had the correct information.

If that information was out of date, perhaps because the human resources data were reloaded from a backup, then the correct telephone number in the LDAP directory will be deleted.

With multi-mater replication, Directory Server can contain master sources of information on more than one server. Multiple masters keep changelogs and can resolve conflicts more safely. A limited number of Directory Server are considered masters which can accept changes; they then replicate the data to replica servers, or consumer servers.[1] Having more than on data master server provides safe failover in the event that a server goes off-line. For more information about replication and multi-master replication, see Chapter 7, *Designing the Replication Process*.

Synchronization allows Directory Server users, groups, attributes, and passwords to be integrated with Microsoft Active Directory users, groups, attributes, and passwords. With two directory services, decide whether they will handle the same information, what amount of that information will be shared, and which service will be the data master for that information. The best course is to choose a single application to master the data and allow the synchronization process to add, update, or delete the entries on the other service.

2.3.6. Determining Data Ownership

Data ownership refers to the person or organization responsible for making sure the data is up-to-date. During the data design phase, decide who can write data to the directory. The following are some common strategies for deciding data ownership:

- Allow read-only access to the directory for everyone except a small group of directory content managers.
- Allow individual users to manage some strategic subset of information for themselves.

  This subset of information might include their passwords, descriptive information about themselves and their role within the organization, their automobile license plate number, and contact information such as telephone numbers or office numbers.
- Allow a person's manager to write to some strategic subset of that person's information, such as contact information or job title.
- Allow an organization's administrator to create and manage entries for that organization.

  This approach allows an organization's administrators to function as the directory content managers.
- Create roles that give groups of people read or write access privileges.

  For example, there can be roles created for human resources, finance, or accounting. Allow each of these roles to have read access, write access, or both to the data needed by the group. This could include salary information, government identification numbers, and home phone numbers and address.

  For more information about roles and grouping entries, see Section 4.3, “Grouping Directory Entries”.

There may be multiple individuals who need to have write access to the same information. For example, an information systems (IS) or directory management group probably requires write access to employee passwords. It may also be desirable for employees themselves to have write access to their own passwords. While, generally, multiple people will have write access to the same information, try to keep this group small and easy to identify. Keeping the group small helps ensure data integrity.

For information on setting access control for the directory, see Chapter 9, *Designing a Secure Directory*.

2.3.7. Determining Data Access

After determining data ownership, decide who can read each piece of data. For example, employees' home phone numbers can be stored in the directory. This data may be useful for a number of organizations, including the employee's manager and human resources. Employees should be able to read this information for verification purposes. However, home contact information can be considered sensitive, so it probably should not be widely available across the enterprise.

For each piece of information stored in the directory, decide the following:

- Can the data be read anonymously?

  The LDAP protocol supports anonymous access and allows easy lookups for common information such as office sites, email addresses, and business telephone numbers. However, anonymous access gives anyone with access to the directory access to the common information. Consequently, use anonymous access sparingly.
- Can the data be read widely across the enterprise?

  Access control can be set so that the client must log into (or bind to) the directory to read specific information. Unlike anonymous access, this form of access control ensures that only members of the organization can view directory information. It also captures login information in the directory's access log so there is a record of who accessed the information.

  For more information about access controls, see Section 9.7, “Designing Access Control”.
- Is there an identifiable group of people or applications that need to read the data?

  Anyone who has write privileges to the data generally also needs read access (with the exception of write access to passwords). There may also be data specific to a particular organization or project group. Identifying these access needs helps determine what groups, roles, and access controls the directory needs.

  For information about groups and roles, see Chapter 4, *Designing the Directory Tree*. For information about access controls, see Section 9.7, “Designing Access Control”.

Making these decisions for each piece of directory data defines a security policy for the directory. These decisions depend upon the nature of the site and the kinds of security already available at the site. For example, having a firewall or no direct access to the Internet means it is safer to support anonymous access than if the directory is placed directly on the Internet. Additionally, some information may only need access controls and authentication measures to restrict access adequately; other sensitive information may need to be encrypted within the database as it is stored.

In many countries, data protection laws govern how enterprises must maintain personal information and restrict who has access to the personal information. For example, the laws may prohibit anonymous access to addresses and phone numbers or may require that users have the ability to view and correct information in entries that represent them. Be sure to check with the organization's legal department to ensure that the directory deployment follows all necessary laws for the countries in which the enterprise operates.

The creation of a security policy and the way it is implemented is described in detail in Chapter 9, *Designing a Secure Directory*.

2.4. DOCUMENTING THE SITE SURVEY

Because of the complexity of data design, document the results of the site surveys. Each step of the site survey can use simple tables to track data. Consider building a master table that outlines the decisions and outstanding concerns. A good tip is to use a spreadsheet so that the table's contents can easily be sorted and searched.

Table 2.4, “Example: Tabulating Data Ownership and Access” identifies data ownership and data access for each piece of data identified by the site survey.

Table 2.4. Example: Tabulating Data Ownership and Access

Data Name	Owner	Supplier Server/Application	Self Read/Write	Global Read	HR Writable	IS Writable
Employee name	HR	PeopleSoft	Read-only	Yes (anonymous)	Yes	Yes
User password	IS	Directory US-1	Read/Write	No	No	Yes
Home phone number	HR	PeopleSoft	Read/Write	No	Yes	No
Employee location	IS	Directory US-1	Read-only	Yes (must log in)	No	Yes
Office phone number	Facilities	Phone switch	Read-only	Yes (anonymous)	No	No

Each row in the table shows what kind of information is being assessed, what departments have an interest in it, and how the information is used and accessed. For example, on the first row, the *employee names* data have the following management considerations:

- *Owner*. Human Resources owns this information and therefore is responsible for updating and changing it.
- *Supplier Server/Application*. The PeopleSoft application manages employee name information.
- *Self Read/Write*. A person can read his own name but not write (or change) it.
- *Global Read*. Employee names can be read anonymously by everyone with access to the directory.
- *HR Writable*. Members of the human resources group can change, add, and delete employee names in the directory.
- *IS Writable*. Members of the information services group can change, add, and delete employee names in the directory.

2.5. REPEATING THE SITE SURVEY

There may need to be more than one site survey, particularly if an enterprise has offices in multiple cities or countries. The informational needs might be so complex that several different organizations have to keep information at their local offices rather than at a single, centralized site.

In this case, each office that keeps a master copy of information should perform its own site survey. After the site survey process has been completed, the results of each survey should be returned to a central team (probably consisting of representatives from each office) for use in the design of the enterprise-wide data schema model and directory tree.

[1] In replication, a *consumer server* or *replica server* is a server that receives updates from a supplier server or hub server.

CHAPTER 3. DESIGNING THE DIRECTORY SCHEMA

The site survey conducted in Chapter 2, *Planning the Directory Data* revealed information about the data which will be stored in the directory. The directory *schema* describes the types of data in the directory, so determining what schema to use reflects decisions on how to represent the data stored in the directory. During the schema design process, each data element is mapped to an LDAP attribute, and related elements are gathered into LDAP object classes. A well-designed schema helps to maintain the integrity of the directory data.

This chapter describes the directory schema and how to design a schema for unique organizational needs.

For information on replicating a schema, see Section 7.4.4, "Schema Replication".

3.1. SCHEMA DESIGN PROCESS OVERVIEW

During the schema design process, select and define the object classes and attributes used to represent the entries stored by Red Hat Directory Server. Schema design involves the following steps:

1. Choosing predefined schema elements to meet as many of data needs as possible.
2. Extending the standard Directory Server schema to define new elements to meet other remaining needs.
3. Planning for schema maintenance.

The simplest and most easily-maintained option is to use existing schema elements defined in the standard schema provided with Directory Server. Choosing standard schema elements helps ensure compatibility with directory-enabled applications. Because the schema is based on the LDAP standard, it has been reviewed and agreed to by a wide number of directory users.

3.2. STANDARD SCHEMA

The directory schema maintains the integrity of the data stored in the directory by imposing constraints on the size, range, and format of data values. The schema reflects decisions about what types of entries the directory contains (like people, devices, and organizations) and the attributes available to each entry.

The predefined schema included with Directory Server contains both the standard LDAP schema as well as additional application-specific schema to support the features of the server. While this schema meets most directory needs, new object classes and attributes can be added to the schema (*extending* the schema) to accommodate the unique needs of the directory. See Section 3.4, "Customizing the Schema" for information on extending the schema.

3.2.1. Schema Format

Directory Server bases its schema format on version 3 of the LDAP protocol. This protocol requires directory servers to publish their schema through LDAP itself, allowing directory client applications to retrieve the schema programmatically and adapt their behavior accordingly. The global set of schema for Directory Server can be found in the **cn=schema** entry.

The Directory Server schema differs slightly from the LDAPv3 schema, because it uses its own proprietary object classes and attributes. In addition, it uses a private field in the schema entries, called **X-ORIGIN**, which describes where the schema entry was defined originally.

For example, if a schema entry is defined in the standard LDAPv3 schema, the **X-ORIGIN** field refers to RFC 2252. If the entry is defined by Red Hat for the Directory Server's use, the **X-ORIGIN** field contains the value **Netscape Directory Server**.

For example, the standard **person** object class appears in the schema as follows:

```
objectclasses: ( 2.5.6.6 NAME 'person' DESC 'Standard Person Object Class' SUP top
    MUST (objectclass $ sn $ cn) MAY (description $ seeAlso $ telephoneNumber $ userPassword)
    X-ORIGIN 'RFC 2252' )
```

This schema entry states the object identifier, or *OID*, for the class (**2.5.6.6**), the name of the object class (**person**), a description of the class (**Standard Person**), and then lists the required attributes (***objectclass***, ***sn***, and ***cn***) and the allowed attributes (***description***, ***seeAlso***, ***telephoneNumber***, and ***userPassword***).

For more information about the LDAPv3 schema format, see the LDAPv3 Attribute Syntax Definitions document, RFC 2252, and other standard schema definitions in RFC 247, RFC 2927, and RFC 2307. All of these schema elements are supported in Red Hat Directory Server.

3.2.2. Standard Attributes

Attributes contain specific data elements such as a name or a fax number. Directory Server represents data as *attribute-data pairs*, a descriptive schema attribute associated with a specific piece of information. These are also called *attribute-value assertions* or AVAs.

For example, the directory can store a piece of data such as a person's name in a pair with the standard attribute, in this case ***commonName*** (***cn***). So, an entry for a person named Babs Jensen has the attribute-data pair **cn: Babs Jensen**.

In fact, the entire entry is represented as a series of attribute-data pairs. The entire entry for Babs Jensen is as follows:

```
dn: uid=bjensen,ou=people,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Babs Jensen
sn: Jensen
givenName: Babs
givenName: Barbara
mail: bjensen@example.com
```

The entry for Babs Jensen contains multiple values for some of the attributes. The ***givenName*** attribute appears twice, each time with a unique value.

In the schema, each attribute definition contains the following information:

- A unique name.
- An object identifier (OID) for the attribute.
- A text description of the attribute.
- The OID of the attribute syntax.
- Indications of whether the attribute is single-valued or multi-valued, whether the attribute is for the directory's own use, the origin of the attribute, and any additional matching rules associated with the attribute.

For example, the ***cn*** attribute definition appears in the schema as follows:

```
attributetypes: ( 2.5.4.3 NAME 'cn' DESC 'commonName Standard Attribute'
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

The attribute's syntax defines the format of the values which the attribute allows. In a way, the syntax helps define the kind of information that can be stored in the attribute. The Directory Server supports all standard attribute syntaxes.

Table 3.1. Supported LDAP Attribute Syntaxes

Name	OID	Definition
Binary	1.3.6.1.4.1.1466.115.121.1.5	For values which are binary.
Bitstring	1.3.6.1.4.1.1466.115.121.1.6	For values which are bitstings. This is supported for searches which require accessing attributes using bitstring flags.
Boolean	1.3.6.1.4.1.1466.115.121.1.7	For attributes with only two allowed values, true or false.
Country String	1.3.6.1.4.1.1466.115.121.1.11	For values which are limited to exactly two printable string characters; for example, US for the United States.
DN	1.3.6.1.4.1.1466.115.121.1.12	For values which are DNs.

Name	OID	Definition
Delivery Method	1.3.6.1.4.1.1466.115.121.1.14	For values which are contained a preferred method of delivering information or contacting an entity. The different values are separated by a dollar sign ($). For example: `telephone $ physical`
DirectoryString (Case-Insensitive String)	1.3.6.1.4.1.1466.115.121.1.15	For values which are case-insensitive strings.
Enhanced Guide	1.3.6.1.4.1.1466.115.121.1.21	For values which contain complex search parameters based on attributes and filters.
Facsimile	1.3.6.1.4.1.1466.115.121.1.22	For values which contain fax numbers.
Fax	1.3.6.1.4.1.1466.115.121.1.23	For values which contain the images of transmitted faxes.
GeneralizedTime	1.3.6.1.4.1.1466.115.121.1.24	For values which are encoded as printable strings. The time zone must be specified. It is strongly recommended to use GMT time.
Guide	1.3.6.1.4.1.1466.115.121.1.25	*Obsolete.* For values which contain complex search parameters based on attributes and filters.
IA5String (Case-Exact String)	1.3.6.1.4.1.1466.115.121.1.26	For values which are case-exact strings.
Integer	1.3.6.1.4.1.1466.115.121.1.27	For values which are whole numbers.
JPEG	1.3.6.1.4.1.1466.115.121.1.28	For values which contain image data.
Name and Optional UID	1.3.6.1.4.1.1466.115.121.1.34	For values which contain a combination value of a DN and (optional) unique ID.
Numeric String	1.3.6.1.4.1.1466.115.121.1.36	For values which contain a string of both numerals and spaces.
OctetString	1.3.6.1.4.1.1466.115.121.1.40	For values which are binary; this is the same as using the binary syntax.
OID	1.3.6.1.4.1.1466.115.121.1.37	For values which contain an object identifier (OID).
Postal Address	1.3.6.1.4.1.1466.115.121.1.41	For values which are encoded in the format **postal-address =** *dstring* *** ("$"** *dstring***)**. For example: `1234 Main St.$Raleigh, NC 12345$USA` Each *dstring* component is encoded as a DirectoryString value. Backslashes and dollar characters, if they occur, are quoted, so that they will not be mistaken for line delimiters. Many servers limit the postal address to 6 lines of up to thirty characters.

Name	OID	Definition
PrintableString	1.3.6.1.4.1.1466.115.121.1.58	For values which contain strings containing alphabetic, numeral, and select punctuation characters (as defined in RFC 4517).
Space-Insensitive String	2.16.840.1.113730.3.7.1	For values which contain space-insensitive strings.
TelephoneNumber	1.3.6.1.4.1.1466.115.121.1.50	For values which are in the form of telephone numbers. It is recommended to use telephone numbers in international form.
Teletex Terminal Identifier	1.3.6.1.4.1.1466.115.121.1.51	For values which contain an international telephone number.
Telex Number	1.3.6.1.4.1.1466.115.121.1.52	For values which contain a telex number, country code, and answerback code of a telex terminal.
URI		For values in the form of a URL, introduced by a string such as **http://**, **https://**, **ftp://**, **ldap://**, and **ldaps://**. The URI has the same behavior as IA5String. See RFC 4517 for more information on this syntax.

3.2.3. Standard Object Classes

Object classes are used to group related information. Typically, an object class represents a real object, such as a person or a fax machine. Before it is possible to use an object class and its attributes in the directory, it must be identified in the schema. The directory recognizes a standard list of object classes by default; these are listed and described in the *Red Hat Directory Server 9 Configuration, Command, and File Reference* .

Each directory entry belongs to at least one object classes. Placing an object class identified in the schema on an entry tells the Directory Server that the entry can have a certain set of possible attribute values and must have another, usually smaller, set of required attribute values.

Object class definitions contain the following information:

- A unique name.
- An *object identifier* (OID) that names the object.
- A set of mandatory attributes.
- A set of allowed (or optional) attributes.

For example, the standard **person** object class appears in the schema as follows:

```
objectclasses: ( 2.5.6.6 NAME 'person' DESC 'Standard Person Object Class' SUP top
    MUST (objectclass $ sn $ cn) MAY (description $ seeAlso $ telephoneNumber $ userPassword)
    X-ORIGIN 'RFC 2252' )
```

As is the case for all of the Directory Server's schema, object classes are defined and stored directly in Directory Server. This means that the directory's schema can be both queried and changed with standard LDAP operations.

3.3. MAPPING THE DATA TO THE DEFAULT SCHEMA

The data identified during the site survey, as described in Section 2.3, “Performing a Site Survey” , must be mapped to the existing default directory schema. This section describes how to view the existing default schema and provides a method for mapping the data to the appropriate existing schema elements.

If there are elements in the schema that do not match the existing default schema, create custom object classes and attributes. See Section 3.4, “Customizing the Schema” for more information.

3.3.1. Viewing the Default Directory Schema

The default directory schema is stored in **/etc/dirsrv/schema/**.

This directory contains all of the common schema for the Directory Server. The LDAPv3 standard user and organization schema can be found in the **00core.ldif** file. The configuration schema used by earlier versions of the directory can be found in the **50ns-directory.ldif** file.

WARNING

Do *not* modify the default directory schema.

For more information about each object class and attribute found in directory, see the *Red Hat Directory Server 9 Configuration, Command, and File Reference*. The same guide also provides more information about the schema files and directory configuration attributes.

3.3.2. Matching Data to Schema Elements

The data identified in the site survey now needs to be mapped to the existing directory schema. This process involves the following steps:

1. Identify the type of object the data describes.

   Select an object that best matches the data described in the site survey. Sometimes, a piece of data can describe multiple objects. Determine if the difference needs to be noted in the directory schema.

   For example, a telephone number can describe an employee's telephone number and a conference room's telephone number. Determine if these different sorts of data need to be considered different objects in the directory schema.

2. Select a similar object class from the default schema.

   It is best to use the common object classes, such as groups, people, and organizations.

3. Select a similar attribute from the matching object class.

   Select an attribute from within the matching object class that best matches the piece of data identified in the site survey.

4. Identify the unmatched data from the site survey.

   If there are some pieces of data that do not match the object classes and attributes defined by the default directory schema, customize the schema. See Section 3.4, “Customizing the Schema” for more information.

For example, the following table maps directory schema elements to the data identified during the site survey in Chapter 2, *Planning the Directory Data*:

Table 3.2. Data Mapped to Default Directory Schema

Data	Owner	Object Class	Attribute
Employee name	HR	person	cn (commonName)
User password	IS	person	userPassword
Home phone number	HR	inetOrgPerson	homePhone
Employee location	IS	inetOrgPerson	localityName
Office phone number	Facilities	person	telephoneNumber

In Table 3.2, “Data Mapped to Default Directory Schema” , the employee name describes a person. In the default directory schema, there is a **person** object class, which inherits from the **top** object class. This object class allows several attributes, one of which is the ***cn*** or ***commonName*** attribute to describe the full name of the person. This

attribute makes the best match for containing the employee name data.

The user password also describes an aspect of the **person** object class, and the ***userPassword*** attribute is listed in the allowed attributes for the **person** object class.

The home phone number describes an aspect of a person; however, there is not a related attribute in the list associated with the **person** object class. The home phone number describes an aspect of a person in an organization's enterprise network. This object corresponds to the **inetOrgPerson** object class in the directory schema. The **inetOrgPerson** object class inherits from the **organizationPerson** object class, which in turn inherits from the **person** object class. Among the **inetOrgPerson** object's allowed attributes is the ***homePhone*** attribute, which is appropriate for containing the employee's home telephone number.

NOTE

The *Red Hat Directory Server 9 Configuration, Command, and File Reference* is useful for determining what attributes are available for your data. Each attribute is listed with object classes which accept it, and each object class is cross-listed with required and allowed attributes.

3.4. CUSTOMIZING THE SCHEMA

The standard schema can be extended if it is too limited for the directory needs. The Directory Server Console can be used to extend the schema by easily adding attributes and object classes. It is also possible to create an LDIF file and add schema elements manually. For more information, see the *Directory Server Administrator's Guide*.

Keep the following rules in mind when customizing the Directory Server schema:

- Keep the schema as simple as possible.
- Reuse existing schema elements whenever possible.
- Minimize the number of mandatory attributes defined for each object class.
- Do not define more than one object class or attribute for the same purpose (data).
- Do not modify any existing definitions of attributes or object classes.

NOTE

When customizing the schema, *never* delete or replace the standard schema. Doing so can lead to compatibility problems with other directories or other LDAP client applications.

Custom object classes and attributes are defined in the **99user.ldif** file. Each individual instance maintains its own **99user.ldif** file in the **/etc/dirsrv/slapd-*instance_name*/schema** directory. It is also possible to create custom schema files and dynamically reload the schema into the server.

3.4.1. When to Extend the Schema

While the object classes and attributes supplied with the Directory Server should meet most common corporate needs, a given object class may not store specialized information about an organization. Also, the schema may need extended to support the object classes and attributes required by an LDAP-enabled application's unique data needs.

3.4.2. Getting and Assigning Object Identifiers

Each LDAP object class or attribute must be assigned a unique name and *object identifier* (OID). When a schema is defined, the elements require a base OID which is unique to your organization. One OID is enough to meet all schema needs. Simply add another level of hierarchy to create new branches for attributes and object classes. Getting and assigning OIDs in schema involves the following steps:

1. Obtain an OID from the Internet Assigned Numbers Authority (IANA) or a national organization.

   In some countries, corporations already have OIDs assigned to them. If your organization does not already have an OID, one can be obtained from IANA. For more information, go to the IANA website at http://www.iana.org/cgi-bin/enterprise.pl.

2. Create an OID registry to track OID assignments.

   An OID registry is a list of the OIDs and descriptions of the OIDs used in the directory schema. This ensures that no OID is ever used for more than one purpose. Then publish the OID registry with the schema.

3. Create branches in the OID tree to accommodate schema elements.

Create at least two branches under the OID branch or the directory schema, using *OID*.**1** for attributes and *OID*.**2** for object classes. To define custom matching rules or controls, add new branches as needed (*OID*.**3**, for example).

3.4.3. Naming Attributes and Object Classes

When creating names for new attributes and object classes, make the names as meaningful as possible. This makes the schema easier to use for Directory Server administrators.

Avoid naming collisions between schema elements and existing schema elements by including a unique prefix on all schema elements. For example, Example Corp. might add the prefix **example** before each of their custom schema elements. They might add a special object class called **examplePerson** to identify Example Corp. employees in their directory.

3.4.4. Strategies for Defining New Object Classes

There are two ways to create new object classes:

- Create many new object classes, one for each object class structure to which to add an attribute.
- Create a single object class that supports all of the custom attributes created for the directory. This kind of object class is created by defining it as an auxiliary object class.

It may be easiest to mix the two methods.

For example, suppose an administrator wants to create the attributes ***exampleDateOfBirth***, ***examplePreferredOS***, ***exampleBuildingFloor***, and ***exampleVicePresident***. A simple solution is to create several object classes that allow some subset of these attributes.

- One object class, **examplePerson**, is created and allows ***exampleDateOfBirth*** and ***examplePreferredOS***. The parent of **examplePerson** is **inetOrgPerson**.
- A second object class, **exampleOrganization**, allows ***exampleBuildingFloor*** and ***exampleVicePresident***. The parent of **exampleOrganization** is the **organization** object class.

The new object classes appear in LDAPv3 schema format as follows:

```
objectclasses: ( 2.16.840.1.117370.999.1.2.3 NAME 'examplePerson' DESC 'Example Person Object Class'
    SUP inetorgPerson MAY (exampleDateOfBirth $ examplePreferredOS) )

objectclasses: ( 2.16.840.1.117370.999.1.2.4 NAME 'exampleOrganization' DESC 'Organization Object Class'
    SUP organization MAY (exampleBuildingFloor $ exampleVicePresident) )
```

Alternatively, create a single object class that allows all of these attributes and use it with any entry which needs these attributes. The single object class appears as follows:

```
objectclasses: (2.16.840.1.117370.999.1.2.5 NAME 'exampleEntry' DESC 'Standard Entry Object Class' SUP top
    AUXILIARY MAY (exampleDateOfBirth $ examplePreferredOS $ exampleBuildingFloor $
exampleVicePresident) )
```

The new **exampleEntry** object class is marked **AUXILIARY**, meaning that it can be used with any entry regardless of its structural object class.

NOTE

The OID of the new object classes in the example (**2.16.840.1.117370**) is based on the former Netscape OID prefix. To create custom object classes, obtain an OID as described in Section 3.4.2, “Getting and Assigning Object Identifiers”.

There are several different ways to organize new object classes, depending on the organization environment. Consider the following when deciding how to implement new object classes:

- Multiple object classes result in more schema elements to create and maintain.

  Generally, the number of elements remains small and needs little maintenance. However, it may be easier to use a single object class if there are more than two or three object classes added to the schema.

- Multiple object classes require a more careful and rigid data design.

Rigid data design forces attention to the object class structure under which every piece of data is placed, which can be either helpful or cumbersome.

- Single object classes simplify data design when there is data that can be applied to more than one type of object class, such as both people and asset entries.

  For example, a custom ***preferredOS*** attribute may be set on both a person and a group entry. A single object class can allow this attribute on both types of entries.

- Avoid required attributes for new object classes.

  Specifying **require** instead of **allow** for attributes in new object classes can make the schema inflexible. When creating a new object class, use **allow** rather than **require** as much as possible.

  After defining a new object class, decide what attributes it allows and requires, and from what object classes it inherits attributes.

3.4.5. Strategies for Defining New Attributes

For both application compatibility and long-term maintenance, try to use standard attributes whenever possible. Search the attributes that already exist in the default directory schema and use them in association with a new object class or check out the *Directory Server Schema Guide* . However, if the standard schema does not contain all the information you need, then add new attributes and new object classes.

For example, a person entry may need more attributes than the **person**, **organizationalPerson**, or **inetOrgPerson** object classes support by default. As an example, no attribute exists within the standard Directory Server schema to store birth dates. A new attribute, ***dateOfBirth***, can be created and set as an allowed attribute within a new auxiliary object class, **examplePerson**.

```
attributetypes: ( dateofbirth-oid NAME 'dateofbirth' DESC 'For employee birthdays'
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'Example defined')

objectclasses: ( 2.16.840.1.117370.999.1.2.3 NAME 'examplePerson' DESC 'Example Person Object Class'
    SUP inetorgPerson MAY (exampleDateOfBirth $ cn) X-ORIGIN 'Example defined')
```

One important thing to remember: *Never* add or delete custom attributes to standard schema elements. If the directory requires custom attributes, add custom object classes to contain them.

3.4.6. Deleting Schema Elements

Do not delete the schema elements included by default with Directory Server. Unused schema elements represent no operational or administrative overhead. Deleting parts of the standard LDAP schema can cause compatibility problems with future installations of Directory Server and other directory-enabled applications.

However, unused custom schema elements can be deleted. Before removing the object class definitions from the schema, modify each entry using the object class. Removing the definition first might prevent the entries that use the object class from being modified later. Schema checks on modified entries also fails unless the unknown object class values are removed from the entry.

3.4.7. Creating Custom Schema Files

Administrators can create custom schema files for the Directory Server to use, in addition to the **99user.ldif** file provided with Directory Server. These schema files hold new, custom attributes and object classes that are specific to the organization. The new schema files should be located in the schema directory, **/etc/dirsrv/schema**.

All standard attributes and object classes are loaded only after custom schema elements have been loaded.

NOTE

Custom schema files should not be numerically or alphabetically higher than **99user.ldif** or the server could experience problems.

After creating custom schema files, there are two ways for the schema changes to be distributed among all servers:

- Manually copy these custom schema files to the instance's schema directory, **/etc/dirsrv/slapd-*instance_name*/schema**. To load the schema, restart the server or reload the schema dynamically by running the **schema-reload.pl** script.

- Modify the schema on the server with an LDAP client such as the Directory Server Console or **ldapmodify**.

- If the server is replicated, then allow the replication process to copy the schema information to each of the consumer servers.

  With replication, all of the replicated schema elements are copied into the consumer servers' **99user.ldif** file. To keep the schema in a custom schema file, like **90example_schema.ldif**, the file has to be copied over to the consumer server manually. Replication does not copy schema files.

If these custom schema files are not copied to all of the servers, the schema information are only replicated to the replica (consumer server) when changes are made to the schema on the supplier server using an LDAP client such as the Directory Server Console or **ldapmodify**.

When the schema definitions are replicated to a consumer server where they do not already exist, they are stored in the **99user.ldif** file. The directory does not track where schema definitions are stored. Storing schema elements in the **99user.ldif** file of consumers does not create a problem as long as the schema is maintained on the supplier server only.

If the custom schema files are copied to each server, changes to the schema files must be copied again to each server. If the files are not copied over again, it is possible the changes will be replicated and stored in the **99user.ldif** file on the consumer. Having the changes in the **99user.ldif** file may make schema management difficult, as some attributes will appear in two separate schema files on a consumer, once in the original custom schema file copied from the supplier and again in the **99user.ldif** file after replication.

For more information about replicating schema, see Section 7.4.4, “Schema Replication”.

3.4.8. Custom Schema Best Practices

When using schema files, be sure to create schema which will be compatible and easy to manage.

3.4.8.1. Naming Schema Files

When naming custom schema files, use the following naming format:

```
[00-99]yourName.ldif
```

Name custom schema files lower (numerically and alphabetically) than **99user.ldif**. This lets Directory Server write to **99user.ldif**, both through LDAP tools and the Directory Server Console.

The **99user.ldif** file contains attributes with an **X-ORIGIN** value of **'user defined'**; however, the Directory Server writes all **'user defined'** schema elements to the highest named file, numerically then alphabetically. If there is a schema file called **99zzz.ldif**, the next time the schema is updated (either through LDAP command-line tools or the Directory Server Console) all of the attributes with an **X-ORIGIN** value of **'user defined'** are written to **99zzz.ldif**. The result is two LDIF files that contain duplicate information, and some information in the **99zzz.ldif** file might be erased.

3.4.8.2. Using 'user defined' as the Origin

Do *not* use **'user defined'** in the **X-ORIGIN** field of custom schema files (such as **60example.ldif**), because **'user defined'** is used internally by the Directory Server when a schema is added over LDAP. In custom schema files, use something more descriptive, such as **'Example Corp. defined'**.

However, if the custom schema elements are added directly to the **99user.ldif** manually, use **'user defined'** as the value of **X-ORIGIN**. If a different **X-ORIGIN** value is set, the server simply may overwrite it.

Using an **X-ORIGIN** of value **'user defined'** ensures that schema definitions in the **99user.ldif** file are not removed from the file by the Directory Server. The Directory Server does not remove them because it relies on an **X-ORIGIN** of value **'user defined'** to tell it what elements should reside in the **99user.ldif** file.

For example:

```
attributetypes: ( exampleContact-oid NAME 'exampleContact'
DESC 'Example Corporate contact'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
X-ORIGIN 'Example defined')
```

After the Directory Server loads the schema entry, it appears as follows:

```
attributetypes: ( exampleContact-oid NAME 'exampleContact'
DESC 'Example Corporate contact'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
```

```
X-ORIGIN ('Example defined' 'user defined') )
```

3.4.8.3. Defining Attributes before Object Classes

When adding new schema elements, all attributes need to be defined before they can be used in an object class. Attributes and object classes can be defined in the same schema file.

3.4.8.4. Defining Schema in a Single File

Each custom attribute or object class should be defined in only one schema file. This prevents the server from overriding any previous definitions when it loads the most recently created schema (as the server loads the schema in numerical order first, then alphabetical order). Decide how to keep from having schema in duplicate files:

- Be careful with what schema elements are included in each schema file.
- Be careful in naming and updating the schema files. When schema elements are edited through LDAP tools, the changes are automatically written to the last file (alphabetically). Most schema changes, then, write to the default file **99user.ldif** and not to the custom schema file, such as **60example.ldif**. Also, the schema elements in **99user.ldif** override duplicate elements in other schema files.
- Add all the schema definitions to the **99user.ldif** file. This is useful if your are managing the schema through the Directory Server Console.

3.5. MAINTAINING CONSISTENT SCHEMA

A consistent schema within Directory Server helps LDAP client applications locate directory entries. Using an inconsistent schema makes it very difficult to efficiently locate information in the directory tree.

Inconsistent schema use different attributes or formats to store the same information. Maintain schema consistency in the following ways:

- Use schema checking to ensure attributes and object classes conform to the schema rules.
- Use syntax validation to enforce attribute values match the required attribute syntax.
- Select and apply a consistent data format.

3.5.1. Schema Checking

Schema checking ensures that all new or modified directory entries conform to the schema rules. When the rules are violated, the directory rejects the requested change.

NOTE

Schema checking checks only that the proper attributes are present. To verify that attribute values are in the correct syntax, use syntax validation, as described in Section 3.5.2, “Syntax Validation”.

By default, the directory enables schema checking. Red Hat recommends not disabling this feature. For information on enabling and disabling schema checking, see the *Directory Server Administrator's Guide* .

With schema checking enabled, be attentive to required and allowed attributes as defined by the object classes. Object class definitions usually contain at least one required attribute and one or more optional attributes. Optional attributes are attributes that can be, but are not required to be, added to the directory entry. Attempting to add an attribute to an entry that is neither required nor allowed according to the entry's object class definition causes the Directory Server to return an object class violation message.

For example, if an entry is defined to use the **organizationalPerson** object class, then the common name (***cn***) and surname (***sn***) attributes are required for the entry. That is, values for these attributes must be set when the entry is created. In addition, there is a long list of attributes that can optionally be used on the entry, including descriptive attributes like ***telephoneNumber***, ***uid***, ***streetAddress***, and ***userPassword***.

3.5.2. Syntax Validation

Syntax validation means that the Directory Server checks that the value of an attribute matches the required syntax for that attribute. For example, syntax validation will confirm that a new ***telephoneNumber*** attribute actually has a valid telephone number for its value.

3.5.2.1. Overview of Syntax Validation

By default, syntax validation is enabled. This is the most basic syntax validation. As with schema checking, this validates any directory modification and rejects changes that violate the syntax rules. Additional settings can be optionally configured so that syntax validation can log warning messages about syntax violations and then either reject the modification or allow the modification process to succeed.

Syntax validation checks LDAP operations where a new attribute value is added, either because a new attribute is added or because an attribute value is changed. Syntax validation does not process existing attributes or attributes added through database operations like replication. Existing attributes can be validated using a special script, **syntax-validate.pl**.

This feature validates all attribute syntaxes, with the exception of binary syntaxes (which cannot be verified) and non-standard syntaxes, which do not have a defined required format. The syntaxes are validated against RFC 4514, except for DNs, which are validated agains the less strict RFC 1779 or RFC 2253. (Strict DN validation can be configured.)

3.5.2.2. Syntax Validation and Other Directory Server Operations

Syntax validation is mainly relevant for standard LDAP operations like creating entries (add) or editing attributes (modify). Validating attribute syntax can impact other Directory Server operations, however.

Database Encryption

For normal LDAP operations, an attribute is encrypted just before the value is written to the database. This means That encryption occurs *after* the attribute syntax is validated.

Encrypted databases (as described in Section 9.8, "Encrypting the Database") can be exported and imported. Normally, it is strongly recommended that these export and import operations are done with the **-E** flag with **db2ldif** and **ldif2db**, which allows syntax validation to occur just fine for the import operation. However, if the encrypted database is exported without using the **-E** flag (which is not supported), then an LDIF with encrypted values is created. When this LDIF is then imported, the encrypted attributes cannot be validated, a warning is logged, and attribute validation is skipped in the imported entry.

Synchronization

There may be differences in the allowed or enforced syntaxes for attributes in Windows Active Directory entries and Red Hat Directory Server entries. In that case, the Active Directory values could not be properly synced over because syntax validation enforces the RFC standards in the Directory Server entries.

Replication

If the Directory Server 9.0 instance is a supplier which replicates its changes to a consumer, then there is no issue with using syntax validation. However, if the supplier in replication is an older version of Directory Server or has syntax validation disabled, then syntax validation should not be used on the 9.0 consumer because the Directory Server 9.0 consumer may reject attribute values that the master allows.

3.5.3. Selecting Consistent Data Formats

LDAP schema allows any data to be placed on any attribute value. However, it is important to store data consistently in the directory tree by selecting a format appropriate for the LDAP client applications and directory users.

With the LDAP protocol and Directory Server, data must be represented in the data formats specified in RFC 2252. For example, the correct LDAP format for telephone numbers is defined in two ITU-T recommendations documents:

- *ITU-T Recommendation E.123*. Notation for national and international telephone numbers.
- *ITU-T Recommendation E.163*. Numbering plan for the international telephone services. For example, a US phone number is formatted as **+1 555 222 1717**.

As another example, the ***postalAddress*** attribute expects an attribute value in the form of a multi-line string that uses dollar signs ($) as line delimiters. A properly formatted directory entry appears as follows:

```
postalAddress: 1206 Directory Drive$Pleasant View, MN$34200
```

Attributes can require strings, binary input, integers, and other formats. The allowed format is set in the schema definition for the attribute.

3.5.4. Maintaining Consistency in Replicated Schema

When the directory schema is edited, the changes are recorded in the changelog. During replication, the changelog is scanned for changes, and any changes are replicated. Maintaining consistency in replicated schema allows replication to continue smoothly. Consider the following points for maintaining consistent schema in a replicated environment:

- Do not modify the schema on a read-only replica.

  Modifying the schema on a read-only replica introduces an inconsistency in the schema and causes replication to fail.

- Do not create two attributes with the same name that use different syntaxes.

  If an attribute is created in a read-write replica that has the same name as an attribute on the supplier replica but has a different syntax from the attribute on the supplier, replication will fail.

3.6. OTHER SCHEMA RESOURCES

See the following links for more information about standard LDAPv3 schema:

- RFC 2251: Lightweight Directory Access Protocol (v3), http://www.ietf.org/rfc/rfc2251.txt
- RFC 2252: LDAPv3 Attribute Syntax Definitions, http://www.ietf.org/rfc/rfc2252.txt
- RFC 2256: Summary of the X.500 User Schema for Use with LDAPv3, http://www.ietf.org/rfc/rfc2256.txt
- Internet Engineering Task Force (IETF), http://www.ietf.org/
- *Understanding and Deploying LDAP Directory Services* . T. Howes, M. Smith, G. Good, Macmillan Technical Publishing, 1999.

CHAPTER 4. DESIGNING THE DIRECTORY TREE

The *directory tree* provides a way to see the data stored by the directory service. The types of information stored in the directory, the physical nature of the enterprise, the applications used with the directory, and the types of replication implemented shape the design of the directory tree.

This chapter outlines the steps for designing the directory tree.

4.1. INTRODUCTION TO THE DIRECTORY TREE

The directory tree provides a means for the directory data to be named and referred to by client applications. The directory tree interacts closely with other design decisions, including the choices available distributing, replicating, or controlling access to the directory data. Invest time to properly design the directory tree before deployment. A properly designed directory tree can save considerable time and effort both during the deployment phase, and later when the directory service is in operation.

A well-designed directory tree provides the following:

- Simplified directory data maintenance.
- Flexibility in creating replication policies and access controls.
- Support for the applications using the directory service.
- Simplified directory navigation for directory users.

The structure of the directory tree follows the hierarchical LDAP model. A directory tree provides a way to organize the data in different logical ways, such as by group, personnel, or place. It also determines how to partition data across multiple servers. For example, each database needs data to be partitioned at the suffix level. Without the proper directory tree structure, it may not be able to spread the data across multiple servers efficiently.

In addition, replication is constrained by the type of directory tree structure used. Carefully define partitions for replication to work. To replicate only portions of the directory tree, take that into account during the design process.

To use access controls on branch points, also consider that in the directory tree design.

NOTE

Directory Server supports a concept for hierarchical navigation and organization of directory information called virtual directory information tree views. See Section 4.4, “Virtual Directory Information Tree Views” before designing the directory tree.

4.2. DESIGNING THE DIRECTORY TREE

There are several major decisions to plan in the directory tree design:

- Choosing a suffix to contain the data.
- Determining the hierarchical relationship among data entries.
- Naming the entries in the directory tree hierarchy.

4.2.1. Choosing a Suffix

The suffix is the name of the entry at the root of the directory tree, and the directory data are stored beneath it. The directory can contain more than one suffix. It is possible to use multiple suffixes if there are two or more directory trees of information that do not have a natural common root.

By default, the standard Directory Server deployment contains multiple suffixes, one for storing data and the others for data needed by internal directory operations (such as configuration information and the directory schema). For more information on these standard directory suffixes, see the *Red Hat Directory Server Administrator's Guide* .

4.2.1.1. Suffix Naming Conventions

All entries in the directory should be located below a common base entry, the *root suffix*. When choosing a name for the root directory suffix, consider these four points to make the name effective:

- Globally unique.
- Static, so it rarely, if ever, changes.

- Short, so that entries beneath it are easier to read on screen.
- Easy for a person to type and remember.

In a single enterprise environment, choose a directory suffix that aligns with a DNS name or Internet domain name of the enterprise. For example, if the enterprise owns the domain name of **example.com**, then the directory suffix is logically **dc=example,dc=com**.

The ***dc*** attribute represents the suffix by breaking the domain name into its component parts.

Normally, any attribute can be used to name the root suffix. However, for a hosting organization, limit the root suffix to the following attributes:

- ***dc*** defines an component of the domain name.
- ***c*** contains the two-digit code representing the country name, as defined by ISO.
- ***l*** identifies the county, city, or other geographical area where the entry is located or that is associated with the entry.
- ***st*** identifies the state or province where the entry resides.
- ***o*** identifies the name of the organization to which the entry belongs.

The presence of these attributes allows for interoperability with subscriber applications. For example, a hosting organization might use these attributes to create a root suffix for one of its clients, **example_a**, such as **o=example_a, st=Washington,c=US**.

Using an organization name followed by a country designation is typical of the X.500 naming convention for suffixes.

4.2.1.2. Naming Multiple Suffixes

Each suffix used with the directory is a unique directory tree. There are several ways to include multiple trees in the directory service. The first is to create multiple directory trees stored in separate databases served by Directory Server.

For example, create separate suffixes for **example_a** and **example_b** and store them in separate databases.

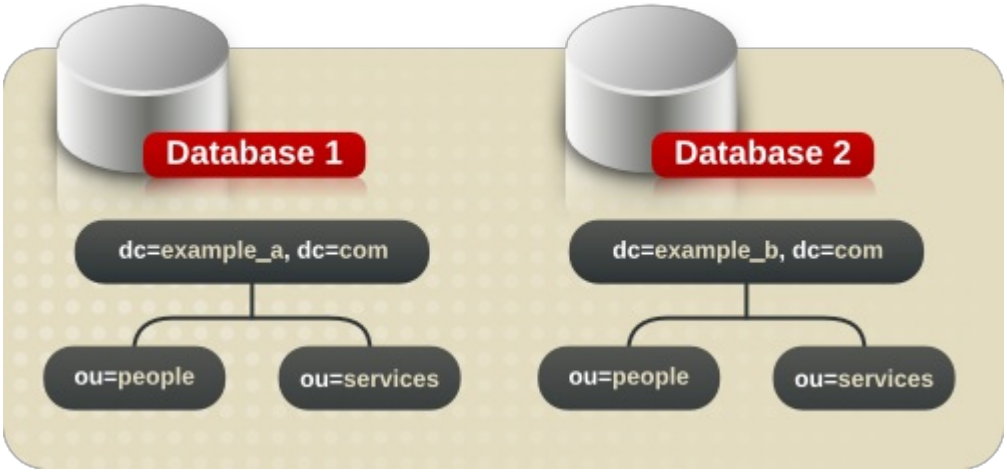


Figure 4.1. Including Multiple Directory Trees in a Database

The databases could be stored on a single server or multiple servers depending on resource constraints.

4.2.2. Creating the Directory Tree Structure

Decide whether to use a flat or a hierarchical tree structure. As a general rule, try to make the directory tree as flat as possible. However, a certain amount of hierarchy can be important later when information is partitioned across multiple databases, prepare replication, and set access controls.

The structure of the tree involves the following steps and considerations:

- Section 4.2.2.1, “Branching the Directory”
- Section 4.2.2.2, “Identifying Branch Points”

- Section 4.2.2.3, “Replication Considerations”
- Section 4.2.2.4, “Access Control Considerations”

4.2.2.1. Branching the Directory

Design the hierarchy to avoid problematic name changes. The flatter a namespace is, the less likely the names are to change. The likelihood of a name changing is roughly proportional to the number of components in the name that can potentially change. The more hierarchical the directory tree, the more components in the names, and the more likely the names are to change.

Following are some guidelines for designing the directory tree hierarchy:

- Branch the tree to represent only the largest organizational subdivisions in the enterprise.

  Any such branch points should be limited to divisions (such as Corporate Information Services, Customer Support, Sales and Professional Services, and Engineering). Make sure that the divisions used to branch the directory tree are stable; do not perform this kind of branching if the enterprise reorganizes frequently.

- Use functional or generic names rather than actual organizational names for the branch points.

  Names change. While subtrees can be renamed, it can be a long and resource-intensive process for large suffixes with many children entries. Using generic names that represent the function of the organization (for example, use **Engineering** instead of **Widget Research and Development**) makes it much less likely that you will need to rename a subtree after organizational or project changes.

- If there are multiple organizations that perform similar functions, try creating a single branch point for that function instead of branching based along divisional lines.

  For example, even if there are multiple marketing organizations, each of which is responsible for a specific product line, create a single **ou=Marketing** subtree. All marketing entries then belong to that tree.

Branching in an Enterprise Environment

Name changes can be avoided if the directory tree structure is based on information that is not likely to change. For example, base the structure on types of objects in the tree rather than organizations. This helps avoid shuffling an entry between organizational units, which requires modifying the distinguished name (DN), which is an expensive operation.

There are a handful of common objects that are good to use to define the structure:

- **ou=people**
- **ou=groups**
- **ou=services**

A directory tree organized using these objects might appear as shown below.

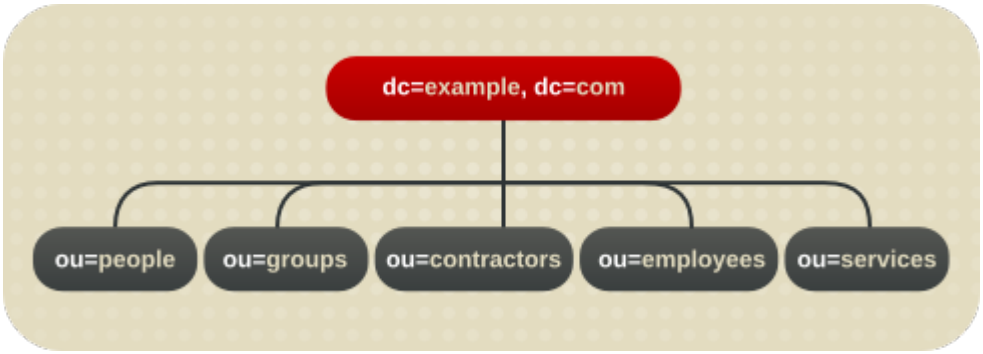


Figure 4.2. Example Environment Directory Tree

Branching in a Hosting Environment

For a hosting environment, create a tree that contains two entries of the object class ***organization*** (***o***) and one entry of the object class ***organizationalUnit*** (***ou***) beneath the root suffix. For example, Example ISP branches their directory as shown below.

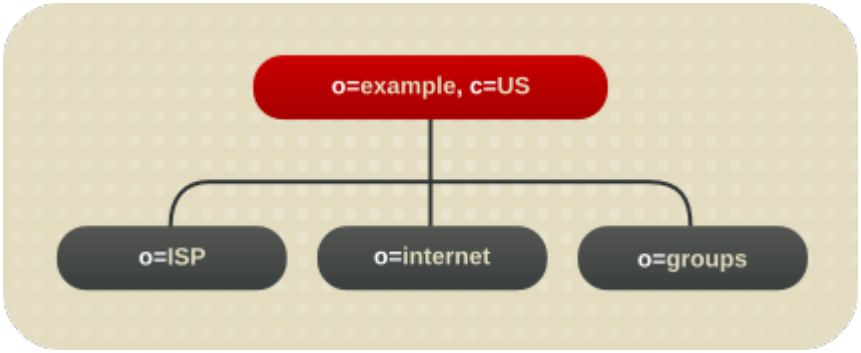


Figure 4.3. Example Hosting Directory Tree

4.2.2.2. Identifying Branch Points

When planning the branches in the directory tree, decide what attributes to use to identify the branch points. Remember that a DN is a unique string composed of attribute-data pairs. For example, the DN of an entry for Barbara Jensen, an employee of Example Corp., is **uid=bjensen,ou=people,dc=example,dc=com**.

Each attribute-data pair represents a branch point in the directory tree, as shown in Figure 4.4, "The Directory Tree for Example Corp." for the directory tree for the enterprise Example Corp.

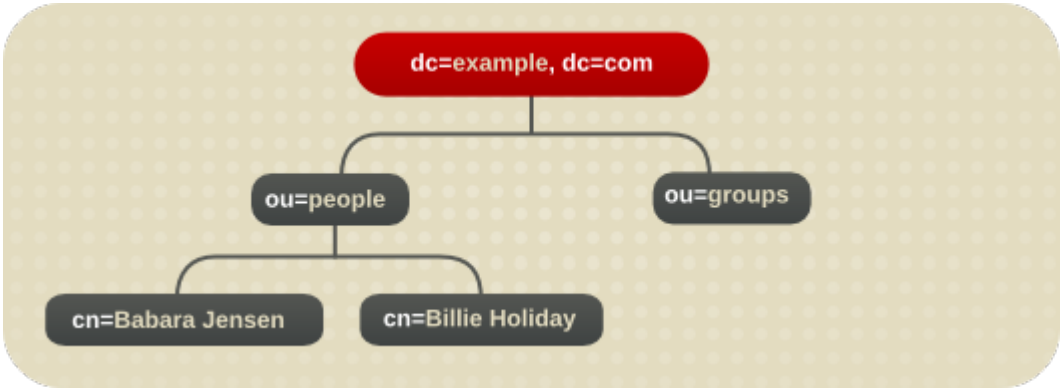


Figure 4.4. The Directory Tree for Example Corp.

Figure 4.5, "Directory Tree for Example ISP" shows the directory tree for Example ISP, an Internet host.

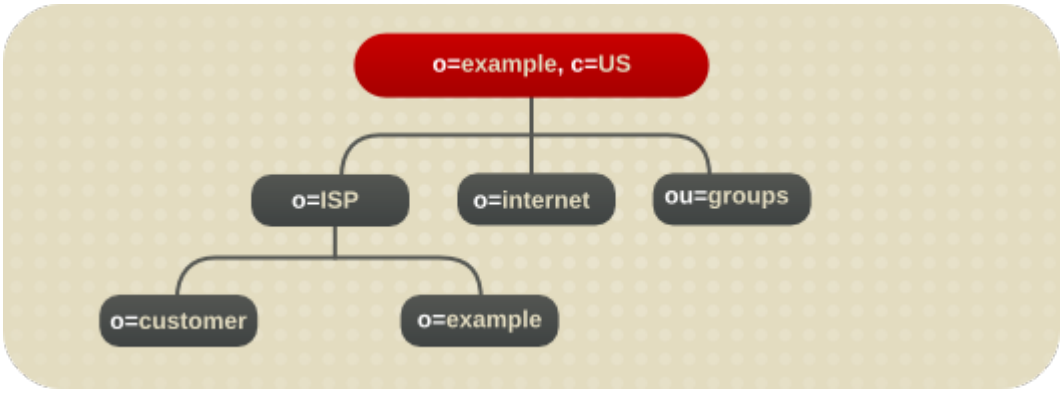


Figure 4.5. Directory Tree for Example ISP

Beneath the suffix entry **c=US,o=example**, the tree is split into three branches. The ISP branch contains customer data and internal information for Example ISP. The internet branch is the domain tree. The groups branch contains information about the administrative groups.

Consider the following when choosing attributes for the branch points:

- Be consistent.

  Some LDAP client applications may be confused if the distinguished name (DN) format is inconsistent across the directory tree. That is, if ***l*** is subordinate to ***ou*** in one part of the directory tree, then make sure ***l*** is subordinate to ***ou*** in all other parts of the directory service.

- Try to use only the traditional attributes (shown in Section 4.2.2.2, "Identifying Branch Points").

  Using traditional attributes increases the likelihood of retaining compatibility with third-party LDAP client applications. Using the traditional attributes also means that they are known to the default directory schema, which makes it easier to build entries for the branch DN.

Table 4.1. Traditional DN Branch Point Attributes

Attribute Name	Definition
dc	An element of the domain name, such as **dc=example**; this is frequently specified in pairs, or even longer, depending on the domain, such as **dc=example,dc=com** or **dc=mtv,dc=example,dc=com**.
c	A country name.
o	An organization name. This attribute is typically used to represent a large divisional branching such as a corporate division, academic discipline (the humanities, the sciences), subsidiary, or other major branching within the enterprise, as in Section 4.2.1.1, "Suffix Naming Conventions".
ou	An organizational unit. This attribute is typically used to represent a smaller divisional branching of the enterprise than an organization. Organizational units are generally subordinate to the preceding organization.
st	A state or province name.
l or ***locality***	A locality, such as a city, country, office, or facility name.

NOTE

A common mistake is to assume that the directory is searched based on the attributes used in the distinguished name. The distinguished name is only a unique identifier for the directory entry and cannot be used as a search key. Instead, search for entries based on the attribute-data pairs stored on the entry itself. Thus, if the distinguished name of an entry is **uid=bjensen,ou=People,dc=example,dc=com**, then a search for **dc=example** does not match that entry unless **dc:example** has explicitly been added as an attribute in that entry.

4.2.2.3. Replication Considerations

During the directory tree design process, consider which entries are being replicated. A natural way to describe a set of entries to be replicated is to specify the DN at the top of a subtree and replicate all entries below it. This subtree also corresponds to a database, a directory partition containing a portion of the directory data.

For example, in an enterprise environment, one method is to organize the directory tree so that it corresponds to the network names in the enterprise. Network names tend not to change, so the directory tree structure is stable. Further, using network names to create the top level branches of the directory tree is useful when using replication to tie together different Directory Servers.

For instance, Example Corp. has three primary networks known as **flightdeck.example.com**, **tickets.example.com**, and **hangar.example.com**. They initially branch their directory tree into three main groups for their major organizational divisions.

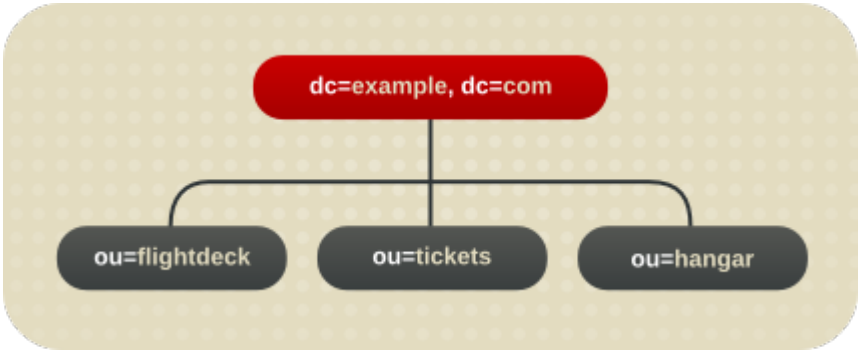


Figure 4.6. Initial Branching of the Directory Tree for Example Corp.

After creating the initial structure of the tree, they create additional branches that show the breakdown of each organizational group.

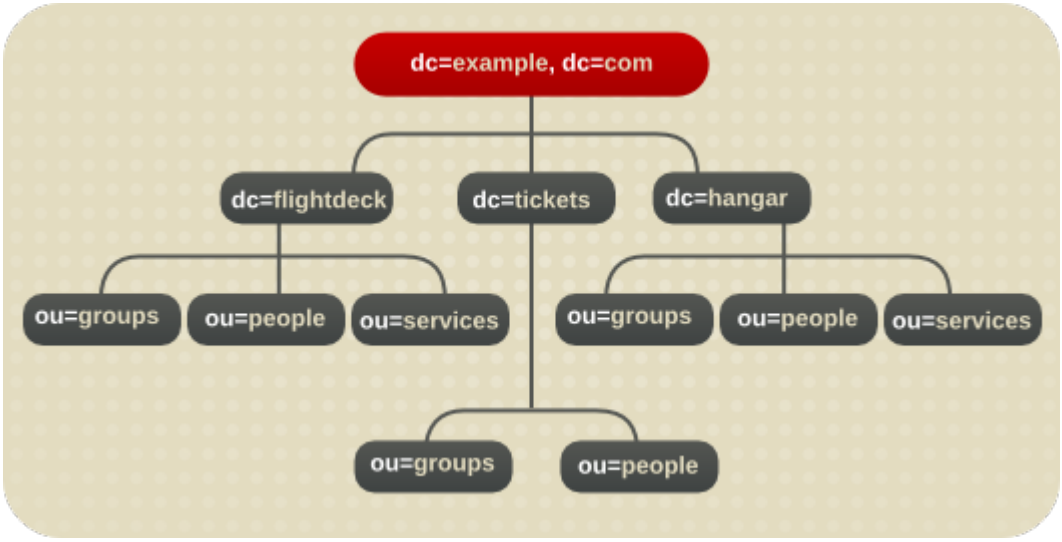


Figure 4.7. Extended Branching for Example Corp.

The Example ISP branches their directory in an asymmetrical tree that mirrors their organization.

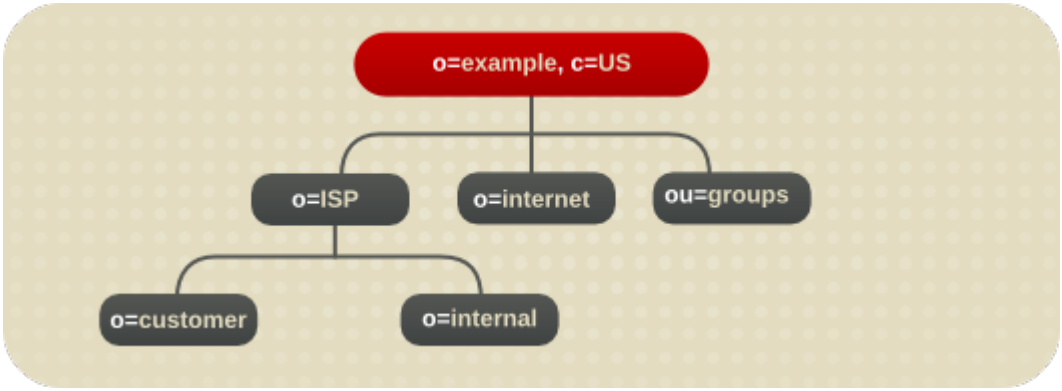


Figure 4.8. Directory Branching for Example ISP

After creating the initial structure of their directory tree, they create additional branches for logical subgroups.

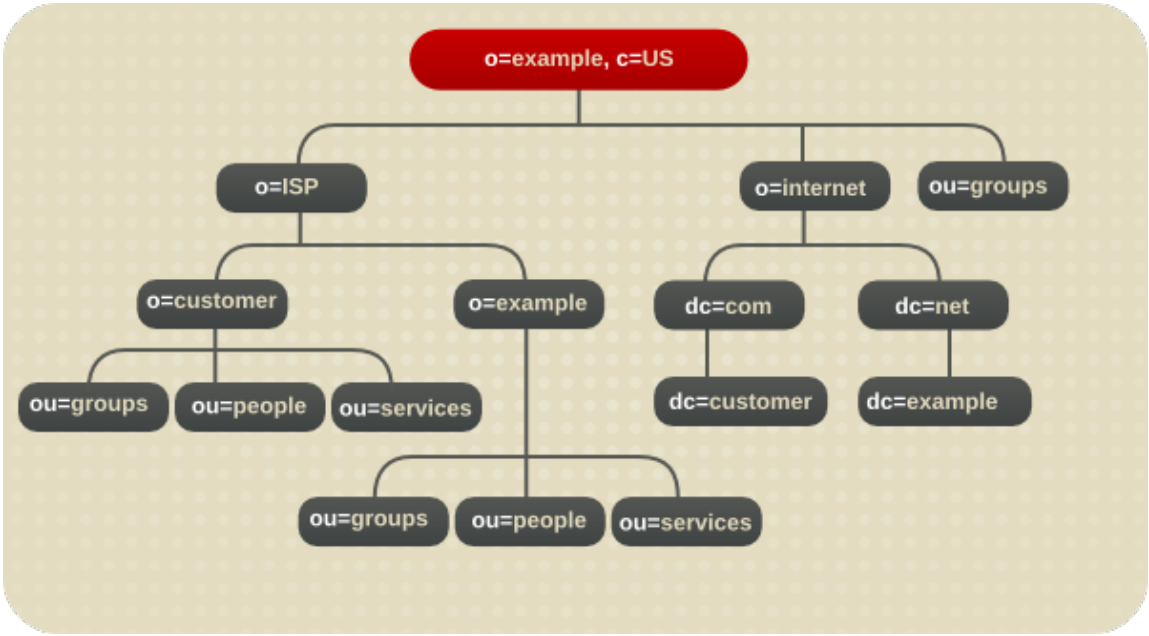


Figure 4.9. Extended Branching for Example ISP

Both the enterprise and the hosting organization design their data hierarchies based on information that is not likely to change often.

4.2.2.4. Access Control Considerations

Introducing a hierarchy into the directory tree can be used to enable certain types of access control. As with replication, it is easier to group similar entries and then administer them from a single branch.

It is also possible to enable the distribution of administration through a hierarchical directory tree. For example, to give an administrator from the marketing department access to the marketing entries and an administrator from the sales department access to the sales entries, design the directory tree according to those divisions.

Access controls can be based on the directory content rather than the directory tree. The filtered mechanism can define a single access control rule stating that a directory entry has access to all entries containing a particular attribute value. For example, set an ACI filter that gives the sales administrator access to all the entries containing the attribute value **ou=Sales**.

However, ACI filters can be difficult to manage. Decide which method of access control is best suited to the directory: organizational branching in the directory tree hierarchy, ACI filters, or a combination of the two.

4.2.3. Naming Entries

After designing the hierarchy of the directory tree, decide which attributes to use when naming the entries within the structure. Generally, names are created by choosing one or more of the attribute values to form a *relative distinguished name (RDN)*. The RDN is a single component within the DN. This is the very first component shown, so the attribute used for that component is the *naming attribute*, because it sets the unique name for the entry. The attributes to use depends on the type of entry being named.

The entry names should adhere to the following rules:

- The attribute selected for naming should be unlikely to change.
- The name must be unique across the directory.

  A unique name ensures that a DN can see at most one entry in the directory.

When creating entries, define the RDN within the entry. By defining at least the RDN within the entry, the entry can be located more easily. This is because searches are not performed against the actual DN but rather the attribute values stored in the entry itself.

Attribute names have a meaning, so try to use the attribute name that matches the type of entry it represents. For example, do not use ***l*** to represent an organization, or ***c*** to represent an organizational unit.

- Section 4.2.3.1, “Naming Person Entries”
- Section 4.2.3.2, “Naming Group Entries”

- Section 4.2.3.3, “Naming Organization Entries”
- Section 4.2.3.4, “Naming Other Kinds of Entries”

4.2.3.1. Naming Person Entries

The person entry's name, the DN, must be unique. Traditionally, distinguished names use the ***commonName***, or ***cn***, attribute to name their person entries. That is, an entry for a person named Babs Jensen might have the distinguished name of **cn=Babs Jensen,dc=example,dc=com**.

While using the common name makes it easier to associated the person with the entry, it might not be unique enough to exclude people with identical names. This quickly leads to a problem known as *DN name collisions*, multiple entries with the same distinguished name.

Avoid common name collisions by adding a unique identifier to the common name, such as **cn=Babs Jensen+employeeNumber=23,dc=example,dc=com**.

However, this can lead to awkward common names for large directories and can be difficult to maintain.

A better method is to identify the person entries with some attribute other than ***cn***. Consider using one of the following attributes:

- ***uid***

  Use the ***uid*** attribute to specify some unique value of the person. Possibilities include a user login ID or an employee number. A subscriber in a hosting environment should be identified by the ***uid*** attribute.

- ***mail***

  The ***mail*** attribute contains a person's email address, which is always unique. This option can lead to awkward DNs that include duplicate attribute values (such as **mail=bjensen@example.com,dc=example,dc=com**), so use this option only if there is not some other unique value to use with the ***uid*** attribute. For example, use the ***mail*** attribute instead of the ***uid*** attribute if the enterprise does not assign employee numbers or user IDs for temporary or contract employees.

- ***employeeNumber***

  For employees of the **inetOrgPerson** object class, consider using an employer assigned attribute value such as ***employeeNumber***.

Whatever is used for an attribute-data pair for person entry RDNs, make sure that they are unique, permanent values. Person entry RDNs should also be readable. For example, **uid=bjensen,dc=example,dc=com** is preferable to **uid=b12r56A,dc=example,dc=com** because recognizable DNs simplify some directory tasks, such as changing directory entries based on their distinguished names. Also, some directory client applications assume that the ***uid*** and ***cn*** attributes use human-readable names.

Considerations for Person Entries in a Hosted Environment

If a person is a subscriber to a service, the entry should be of object class **inetUser**, and the entry should contain the ***uid*** attribute. The attribute must be unique within a customer subtree.

If a person is part of the hosting organization, represent them as an **inetOrgPerson** with the **nsManagedPerson** object class.

Placing Person Entries in the DIT

The following are some guidelines for placing person entries in the directory tree:

- People in an enterprise should be located in the directory tree below the organization's entry.
- Subscribers to a hosting organization need to be below the **ou=people** branch for the hosted organization.

4.2.3.2. Naming Group Entries

There are four main ways to represent a group:

- A *static group* explicitly defines is members. The **groupOfNames** or **groupOfUniqueNames** object classes contain values naming the members of the group. Static groups are suitable for groups with few members, such as the group of directory administrators. Static groups are not suitable for groups with thousands of members.

Static group entries must contain a **uniqueMember** attribute value because **uniqueMember** is a mandatory attribute of the **groupOfUniqueNames** object. This object class requires the ***cn*** attribute, which can be used to form the DN of the group entry.

- A *dynamic group* uses an entry representing the group with a search filter and subtree. Entries matching the filter are members of the group.
- *Roles* unify the static and dynamic group concept. See Section 4.3, "Grouping Directory Entries" for more information.

In a deployment containing hosted organizations, consider using the **groupOfUniqueNames** object class to contain the values naming the members of groups used in directory administration. In a hosted organization, we also recommend that group entries used for directory administration be located under the **ou=Groups** branch.

4.2.3.3. Naming Organization Entries

The organization entry name, like other entry names, must be unique. Using the legal name of the organization along with other attribute values helps ensure the name is unique, such as **o=example_a+st=Washington,o=ISP,c=US**.

Trademarks can also be used, but they are not guaranteed to be unique.

In a hosting environment, use the ***organization*** (***o***) attribute as the naming attribute.

4.2.3.4. Naming Other Kinds of Entries

The directory contains entries that represent many things, such as localities, states, countries, devices, servers, network information, and other kinds of data.

For these types of entries, use the ***cn*** attribute in the RDN if possible. Then, for naming a group entry, name it something like **cn=administrators,dc=example,dc=com**.

However, sometimes an entry's object class does not support the ***commonName*** attribute. Instead, use an attribute that is supported by the entry's object class.

The attributes used for the entry's DN do not have to correspond to the attributes actually used in the entry. However, having some correlation between the DN attributes and attributes used by the entry simplifies administration of the directory tree.

4.2.4. Renaming Entries and Subtrees

Section 4.2.3, "Naming Entries" talks about the importance of naming entries in Red Hat Directory Server. The entry names, in a sense, define the directory tree structure. Each branch point (each entry which has entries beneath it) creates a new link in the hierarchy.

Example 4.1. Building Entry DNs

```
dc=example,dc=com  =>  root suffix
ou=People,dc=example,dc=com  =>  org unit
st=California,ou=People,dc=example,dc=com  =>  state/province
l=Mountain View,st=California,ou=People,dc=example,dc=com  =>  city
ou=Engineering,l=Mountain View,st=California,ou=People,dc=example,dc=com  =>  org unit
uid=jsmith,ou=Engineering,l=Mountain View,st=California,ou=People,dc=example,dc=com  =>  leaf entry
```

When the naming attribute of an entry, the leftmost element of the DN, is changed, this is a *modrdn operation*. That's a special kind of modify operation because, in a sense, it moves the entry within the directory tree. For leaf entries (entries with no children), modrdn operations are lateral moves; the entry has the same parent, just a new name.

Figure 4.10. modrdn Operations for a Leaf Entry

For subtree entries, the modrdn operation not only renames the subtree entry itself, but also changes the DN components of all of the children entries *beneath* the subtree.

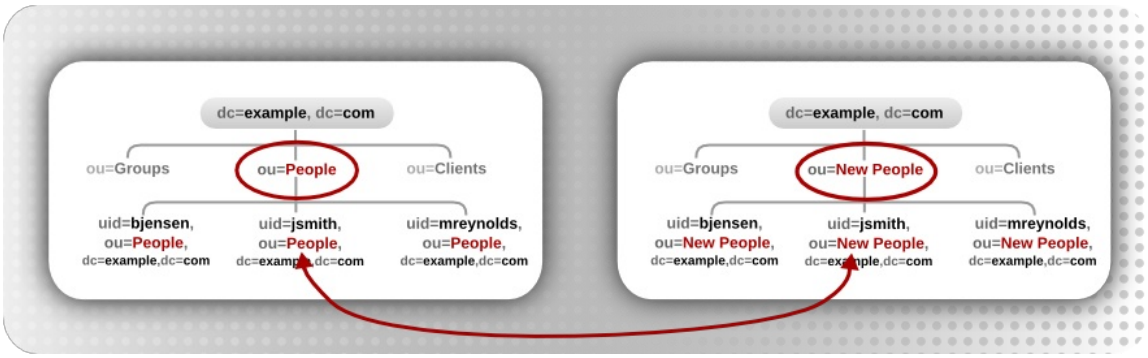


Figure 4.11. modrdn Operations for a Subtree Entry

IMPORTANT

Subtree modrdn operations also move and rename all of the child entries beneath the subtree entry. For large subtrees, this can be a time- and resource-intensive process. Plan the naming structure of your directory tree hierarchy so that it will not require frequent subtree rename operations.

A similar action to renaming a subtree is moving an entry from one subtree to another. This is an expanded type of modrdn operation, which simultaneously renames the entry (even if it is the same name) and sets a ***newsuperior*** attribute which moves the entry from one parent to another.

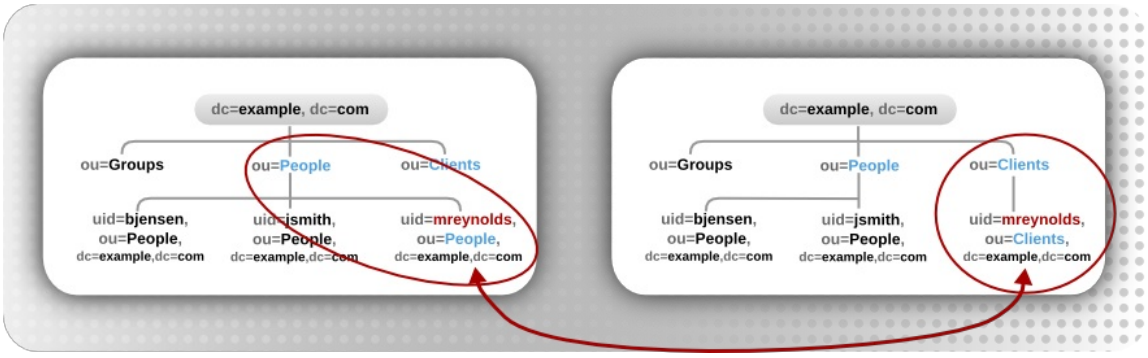


Figure 4.12. modrdn Operations to a New Parent Entry

Both new superior and subtree rename operations are possible because of how entries are stored in the **entryrdn.db4** index. Each entry is identified by its own key (a *self-link*) and then a subkey which identifies its parent (the *parent link*) and any children. This has a format that lays out the directory tree hierarchy by treating parents and children as attribute to an entry, and every entry is describes by a unique ID and its RDN, rather than the full DN.

```
numeric_id:RDN => self link
    ID: #; RDN: "rdn"; NRDN: normalized_rdn
P#:RDN => parent link
    ID: #; RDN: "rdn"; NRDN: normalized_rdn
C#:RDN => child link
    ID: #; RDN: "rdn"; NRDN: normalized_rdn
```

For example, the **ou=people** subtree has a parent of **dc=example,dc=com** and a child of **uid=jsmith**.

```
4:ou=people
  ID: 4; RDN: "ou=People"; NRDN: "ou=people"
P4:ou=people
  ID: 1; RDN: "dc=example,dc=com"; NRDN: "dc=example,dc=com"
C4:ou=people
  ID: 10; RDN: "uid=jsmith"; NRDN: "uid=jsmith"
```

There are some things to keep in mind when performing rename operations:

- You cannot rename the root suffix.
- Subtree rename operations have minimal effect on replication. Replication agreements are applied to an entire database, not a subtree within the database, so a subtree rename operation does not require re-configuring a replication agreement. All of the name changes after a subtree rename operation are replicated as normal.
- Renaming a subtree **may** require any synchronization agreements to be re-configured. Sync agreements are set at the suffix or subtree level, so renaming a subtree may break synchronization.
- Renaming a subtree **requires** that any subtree-level ACIs set for the subtree be re-configured manually, as well as any entry-level ACIs set for child entries of the subtree.
- You can rename a subtree with children, but you cannot delete a subtree with children.
- Trying to change the component of a subtree, like moving from ***ou*** to ***dc***, may fail with a schema violation. For example, the **organizationalUnit** object class requires the ***ou*** attribute. If that attribute is removed as part of renaming the subtree, then the operation will fail.

4.3. GROUPING DIRECTORY ENTRIES

After creating the required entries, group them for ease of administration. The Directory Server supports several methods for grouping entries:

- Using groups
- Using roles

4.3.1. About Groups

Groups, as the name implies, are simply collections of users. There are several different types of groups in Directory Server which reflects the type of memberships allowed, like certificate groups, URL groups, and unique groups (where every member must unique). Each type of group is defined by an object class (such as **groupOfUniqueNames**) and a corresponding member attribute (such as **uniqueMember**).

The type of group identifies the type of members. The configuration of the group depends on how those members are added to the group. Directory Server has two kinds of groups:

- *Static groups* have a finite and defined list of members which are added manually to the group entry.
- *Dynamic groups* use filters to recognize which entries are members of the group, so the group membership is constantly changed as the entries which match the group filter change.

Groups are the simplest form of organizing entries in Directory Server. They are largely manually configured and there is no functionality or behavior for them beyond being an organization method. (Really, groups do not "do" anything to directory entries, though groups can be manipulated by LDAP clients to perform operations.)

4.3.1.1. Listing Group Membership in User Entries

Groups are essentially lists of user DNs. By default, group membership is only reflected in the group entry itself, not on the user entries. The MemberOf Plug-in, however, uses the group member entries to update user entries dynamically, to reflect *on the user entry* what groups the user belongs to. The MemberOf Plug-in automatically scans group entries with a specified member attribute, traces back all of the user DNs, and creates a corresponding ***memberOf*** attribute on the user entry, with the name of the group.

Group membership is *determined* by the member attribute on the group entry, but group membership for all groups for a user is *reflected* in the user's entry in the ***memberOf*** attribute. The name of every group to which a user belongs is listed as a ***memberOf*** attribute. The values of those ***memberOf*** attributes are managed by the Directory Server.

NOTE

It is possible, as outlined in Section 6.2.1, "About Using Multiple Databases" , to store different suffixes in different databases.

By default, the MemberOf Plug-in only looks for potential members for users who are in the same database as the group. If users are stored in a different database than the group, then the user entries will not be updated with ***memberOf*** attributes because the plug-in cannot ascertain the relationship between them.

The MemberOf Plug-in can be configured to search through all configured databases by enabling the ***memberOfAllBackends*** attributes.

A single instance of the MemberOf Plug-in can be configured to identify multiple member attributes by setting the multi-valued ***memberofgroupattr*** in the plug-in entry, so the MemberOf Plug-in can manage multiple types of groups.

4.3.1.2. Automatically Adding New Entries to Groups

Group management can be a critical factor for managing directory data, especially for clients which use Directory Server data and organization or which use groups to apply functionality to entries. Groups make it easier to apply policies consistently and reliably across the directory. Password policies, access control lists, and other rules can all be based on group membership.

Being able to assign new entries to groups, automatically, at the time that an account is created ensures that the appropriate policies and functionality are immediately applied to those entries — without requiring administrator intervention.

The *Automembership Plug-in* essentially allows a static group to act like a dynamic group. It uses a set of rules (based on entry attributes, directory location, and regular expressions) to assign a user automatically to a specified group.

There can be instances where entries that match the LDAP search filter should be added to different groups, depending on the value of some other attribute. For example, machines may need to be added to different groups depending on their IP address or physical location; users may need to be in different groups depending on their employee ID number.

Automember definitions are a set of nested entries, with the Auto Membership Plug-in container, then the automember definition, and then any regular expression conditions for that definition.

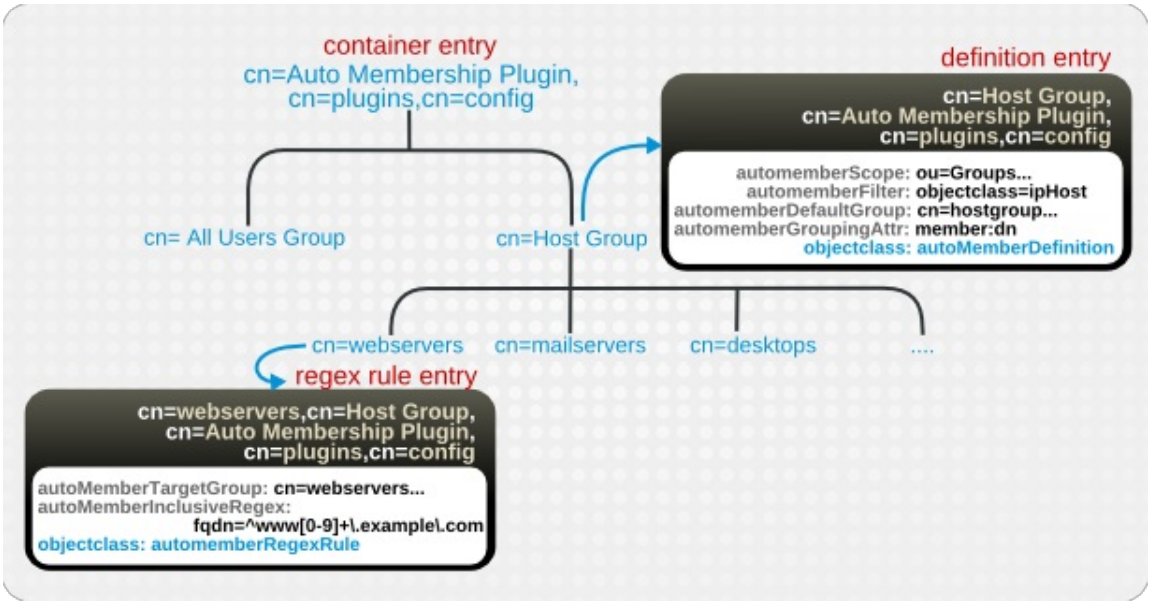


Figure 4.13. Regular Expression Conditions

NOTE

Automembership assignments are only made automatically when an entry is added to the Directory Server.

For existing entries or entries which are edited to meet an automember rule, there is a fix-up task which can be run to assign the proper group membership.

4.3.2. About Roles

Roles are a sort of hybrid group, behaving as both a static and a dynamic group. With a group, entries are added to a group entry as members. With a role, the role attribute is added to an entry and then that attribute is used to identify members in the role entry automatically.

Roles effectively *and automatically* organize users in a number of different ways:

- *Explicitly listing role members.* Viewing the role will display the complete list of members for that role. The role itself can be queried to check membership (which is not possible with a dynamic group).
- *Showing to what roles an entry belongs.* Because role membership is determined by an attribute on an entry, simply viewing an entry will show all of the roles to which it belongs. This is similar to the *memberOf* attributes for groups, only it is not necessary to enable or configure a plug-in instance for this functionality to work. It is automatic.
- *Assigning the appropriate roles.* Role membership is assigned through the *entry*, not through the role, so the roles to which a user belongs can be easily assigned and removed by editing the entry, in a single step.

Managed roles can do everything that can normally be done with static groups. The role members can be filtered using filtered roles, similarly to the filtering with dynamic groups. Roles are easier to use than groups, more flexible in their implementation, and reduce client complexity.

Role *members* are entries that possess the role. Members can be specified either explicitly or dynamically. How role membership is specified depends upon the type of role. Directory Server supports three types of roles:

- *Managed roles* have an explicit enumerated list of members.
- *Filtered roles* are assigned entries to the role depending upon the attribute contained by each entry, specified in an LDAP filter. Entries that match the filter possess the role.
- *Nested roles* are roles that contain other roles.

Roles The concept of activating/inactivating roles allows entire groups of entries to be activated or inactivated in just one operation. That is, the members of a role can be temporarily disabled by inactivating the role to which they belong.

When a role is inactivated, it does not mean that the user cannot bind to the server using that role entry. The meaning of an inactivated role is that the user cannot bind to the server using any of the entries that belong to that role; the entries that belong to an inactivated role will have the ***nsAccountLock*** attribute set to **true**.

When a nested role is inactivated, a user cannot bind to the server if it is a member of any role within the nested role. All the entries that belong to a role that directly or indirectly are members of the nested role have ***nsAccountLock*** set to **true**. There can be several layers of nested roles, and inactivating a nested role at any point in the nesting will inactivate all roles and users beneath it.

4.3.3. Deciding Between Roles and Groups

Roles and groups can accomplish the same goals. Managed roles can do everything that static groups can do, while filtered roles can filter and identify members as dynamic groups do. Both roles and groups have advantages and disadvantages. Deciding whether to use roles or groups (or a mix) depends on balancing client needs and server resources.

Roles reduce client-side complexity, which is their key benefit. With roles, the client application can check role membership by searching the ***nsRole*** operational attribute on entries; this multi-valued attribute identifies every role to which the entry belongs. From the client application point of view, the method for checking membership is uniform and is performed on the server side.

However, this ease of use for clients comes at the cost of increased server complexity. Evaluating roles is more resource-intensive for the Directory Server than evaluating groups because the server does the work for the client application.

While groups are easier for the server, they require smarter and more complex clients to use them effectively. For example, dynamic groups, from an application point of view, offer no support from the server to provide a list of group members. Instead, the application retrieves the group definitions and then runs the filter. Group membership is only reflected on user entries if the appropriate plug-ins are configured. Ultimately, the methods for determining group membership are not uniform or predictable.

NOTE

One thing that can balance managing group membership is the MemberOf Plug-in. Using the ***memberOf*** strikes a nice balance between being simple for the client to use and being efficient for the server to calculate.

The MemberOf Plug-in dynamically creates ***memberOf*** attribute on a user entry whenever a user is added to a group. A client can run a single search on a group entry to get a list of all of its members, or a single search on a user entry to get a complete list of all the groups it belongs to.

The server only has maintenance overhead when the membership is modified. Since both the specified member (group) and ***memberOf*** (user) attributes are stored in the database, there is no extra processing required for searches, which makes the searches from the clients very efficient.

4.4. VIRTUAL DIRECTORY INFORMATION TREE VIEWS

Directory Server supports a concept for hierarchical navigation and organization of directory information called *virtual directory information tree views* or *virtual DIT views*.

NOTE

Virtual views are not entirely compatible with multiple back ends in that the entries to be returned by the views must reside in the same back end; the search is limited to one back end.

4.4.1. About Virtual DIT Views

There are two ways to configure the directory namespace:

- A hierarchical directory information tree.
- A flat directory information tree.

The hierarchical DIT is useful for navigating the directory but is cumbersome and time-consuming to change. A major organizational change to a hierarchical DIT can be an expensive and time-consuming operation, because it usually involves considerable service disruption. This can usually only be minimized by performing changes after hours and during periods of low traffic.

The flat DIT, while requiring little to no change, does not provide a convenient way to navigate or manage the entries in the directory service. A flat DIT also presents many management challenges as administration becomes more complex without any natural hierarchical groupings.

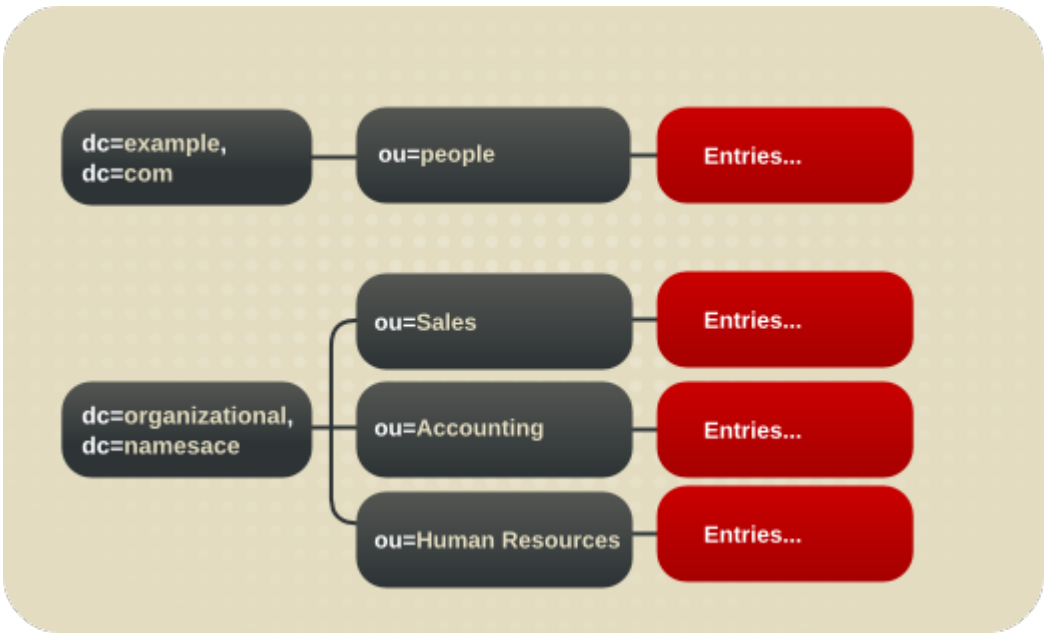


Figure 4.14. Examples of a Flat and an Organizationally-Based DIT

Using a hierarchical DIT, a deployment must then determine the subject domain of the hierarchy. Only one choice can be made; the natural tendency is to choose the organizational hierarchy.

This view of the organization serves well in many cases, but having only a single view can be very limiting for directory

navigation and management. For example, an organizational hierarchy is fine for looking for entries that belong to people in the Accounts department. However, this view is much less useful for finding entries that belong to people in a geographical location, such as Mountain View, California. The second query is as valid as the first, yet it requires knowledge of the attributes contained in the entries and additional search tools. For such a case, navigation using the DIT is not an option.

Similarly, management of the directory is much easier when the DIT matches the requirements of the management function. The organization of the DIT may also be affected by other factors, such as replication and migration considerations, that cause the DIT to have functional utility for those applications but very little practical utility in other cases.

Clearly, hierarchies are a useful mechanism for navigation and management. To avoid the burden of making changes to an existing DIT, however, a deployment may elect to forgo a hierarchy altogether in favor of a flat DIT.

It would be advantageous for deployments if the directory provided a way to create an arbitrary number of hierarchies that get mapped to entries without having to move the target entries in question. The *virtual DIT views* feature of Directory Server resolves the quandary of deciding the type of DIT to use for the directory deployment.

Virtual DIT views provide a way to hierarchically navigate entries without the requirement that those entries physically exist in any particular place. The virtual DIT view uses information about the entries to place them in the view hierarchy. To client applications, virtual DIT views appear as ordinary container hierarchies. In a sense, virtual DIT views superimpose a DIT hierarchy over a set of entries, irrespective of whether those entries are in a flat namespace or in another hierarchy of their own.

Create a virtual DIT view hierarchy in the same way as a normal DIT hierarchy. Create the same entries (for example, organizational unit entries) but with an additional object class (**nsview**) and a filter attribute (***nsviewfilter***) that describes the view. After adding the additional attribute, the entries that match the view filter instantly populate the view. The target entries only *appear* to exist in the view; their true location never changes. Virtual DIT views behave like normal DITs in that a subtree or a one-level search can be performed with the expected results being returned.

For information about adding and modifying entries, see "Creating Directory Entries" in the *Red Hat Directory Server Administrator's Guide*

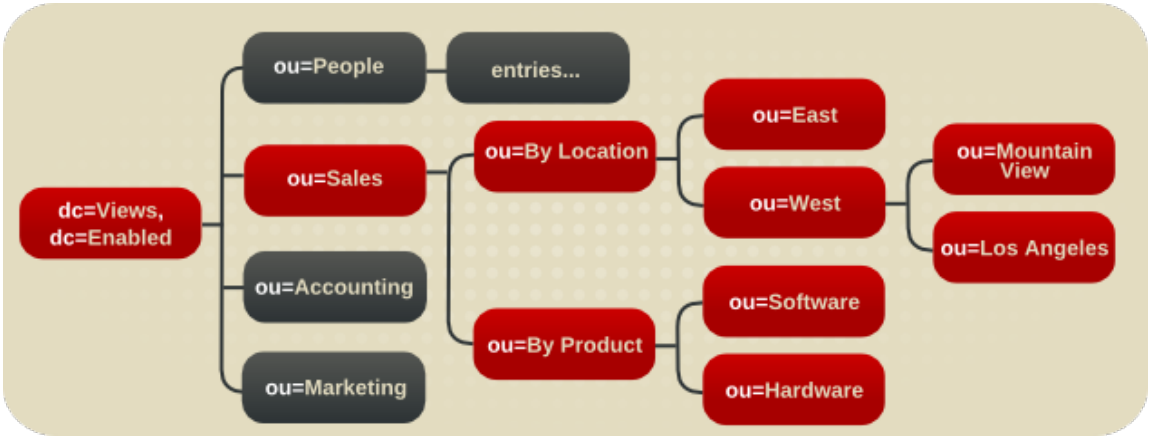


Figure 4.15. A Combined DIT Using Views

The DIT Figure 4.15, "A Combined DIT Using Views" in illustrates what happens when the two DITs shown in Figure 4.14, "Examples of a Flat and an Organizationally-Based DIT" are combined using views. Because views inherently allow entries to appear in more than one place in a view hierarchy, this feature has been used to expand the **ou=Sales** entry to enable viewing the Sales entries either by location or by product.

Given a set of virtual DIT view hierarchies, a directory user can use the view that makes the most sense to navigate to the required entries. For example, if the target entries were those who live in Mountain View, a view which begins by navigating using location-based information is most appropriate. If it were an organizational question, the organization view would be a better choice. Both of these views exist in the Directory Server at the same time and operate on the same entries; the different views just have different objectives when displaying their version of the directory structure.

The entries in the views-enabled directory in Figure 4.15, "A Combined DIT Using Views" are contained in a flat namespace just below the parent of the top-most view in the hierarchy. This is not required. The entries can exist in a hierarchy of their own. The only concern that a view has about the placement of an entry is that it must be a descendant of the parent of the view hierarchy.

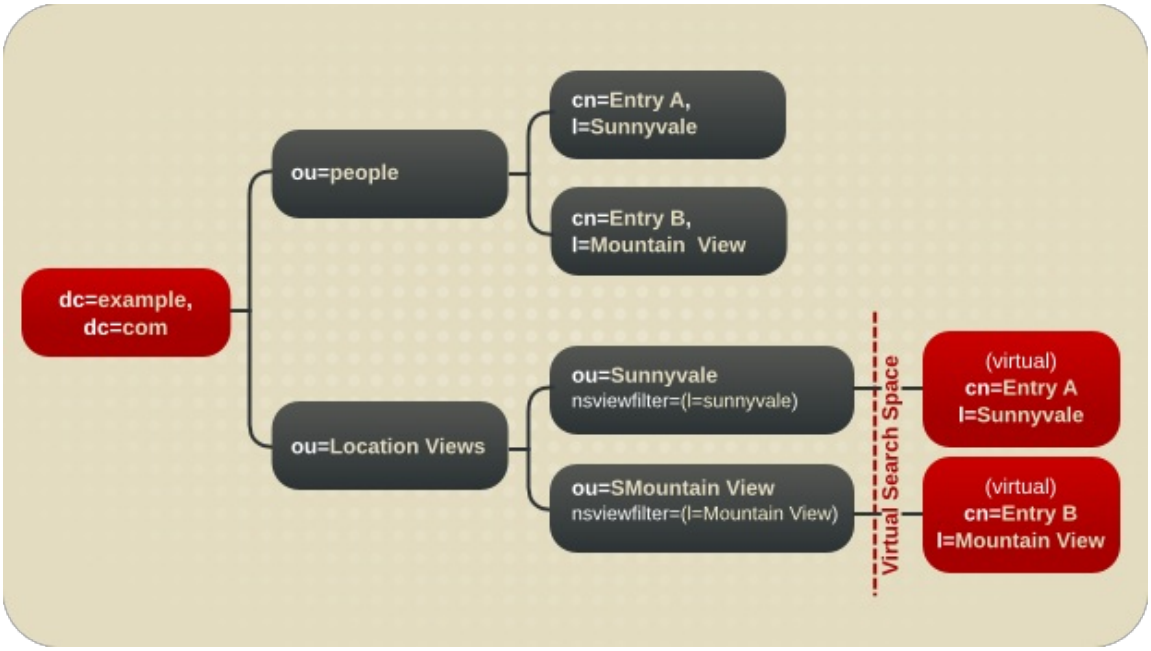


Figure 4.16. A DIT with a Virtual DIT View Hierarchy

- The sub-tree **ou=People** contains the real **Entry A** and **Entry B** entries.
- The sub-tree **ou=Location Views** is a view hierarchy.
- The leaf nodes **ou=Sunnyvale** and **ou=Mountain View** each contain an attribute, ***nsviewfilter***, which describes the view.

  These are leaf nodes because they do not contain the real entries. However, when a client application searches these views, it finds **Entry A** under **ou=Sunnyvale** and **Entry B** under **ou=Mountain View**. This virtual search space is described by the ***nsviewfilter*** attributes of all ancestor views. A search made from a view returns both entries from the virtual search space and those from the actual search space. This enables the view hierarchies to function as a conventional DIT or change a conventional DIT into a view hierarchy.

4.4.2. Advantages of Using Virtual DIT Views

The deployment decisions become easier with virtual DIT views because:

- Views facilitate the use of a flat namespace for entries, because virtual DIT views provide navigational and managerial support similar to those provided by traditional hierarchies.

  In addition, whenever there is a change to the DIT, the entries never need to be moved; only the virtual DIT view hierarchies change. Because these hierarchies contain no real entries, they are simple and quick to modify.
- Oversights during deployment planning are less catastrophic with virtual DIT views. If the hierarchy is not developed correctly in the first instance, it can be changed easily and quickly without disrupting the service.
- View hierarchies can be completely revised in minutes and the results instantly realized, significantly reducing the cost of directory maintenance.

  Changes to a virtual DIT hierarchy are instantly realized. When an organizational change occurs, a new virtual DIT view can be created quickly. The new virtual DIT view can exist at the same time as the old view, thereby facilitating a more gradual changeover for the entries themselves and for the applications that use them. Because an organizational change in the directory is not an all-or-nothing operation, it can be performed over a period of time and without service disruption.
- Using multiple virtual DIT views for navigation and management allows for more flexible use of the directory service.

  With the functionality provided by virtual DIT views, an organization can use both the old and new methods to organize directory data without any requirement to place entries at certain points in the DIT.
- Virtual DIT view hierarchies can be created as a kind of ready-made query to facilitate the retrieval of commonly-required information.

- Views promote flexibility in working practices and reduce the requirement that directory users create complex search filters, using attribute names and values that they would otherwise have no need to know.

  The flexibility of having more than one way to view and query directory information allows end users and applications to find what they need intuitively through hierarchical navigation.

4.4.3. Example of Virtual DIT Views

The LDIF entries below show a virtual DIT view hierarchy that is based on location. Any entry that resides below **dc=example,dc=com** and fits the view description appears in this view, organized by location.

```
dn: ou=Location Views,dc=example,dc=com
objectclass: top
objectclass: organizationalUnit
objectclass: nsView
ou: Location Views
description: views categorized by location


dn: ou=Sunnyvale,ou=Location Views,dc=example,dc=com
objectclass: top
objectclass: organizationalUnit
objectclass: nsView
ou: Sunnyvale
nsViewFilter: (l=Sunnyvale)
description: views categorized by location


dn: ou=Santa Clara,ou=Location Views,dc=example,dc=com
objectclass: top
objectclass: organizationalUnit
objectclass: nsView
ou: Santa Clara
nsViewFilter: (l=Santa Clara)
description: views categorized by location


dn: ou=Cupertino,ou=Location Views,dc=example,dc=com
objectclass: top
objectclass: organizationalUnit
objectclass: nsView
ou: Cupertino
nsViewFilter: (l=Cupertino)
description: views categorized by location
```

A subtree search based at **ou=Location Views,dc=example,dc=com** would return all entries below **dc=example,dc=com** which match the filters **(l=Sunnyvale)**, **(l=Santa Clara)**, or **(l=Cupertino)**. Conversely, a one-level search would return no entries other than the child view entries because all qualifying entries reside in the three descendant views.

The **ou=Location Views,dc=example,dc=com** view entry itself does not contain a filter. This feature facilitates hierarchical organization without the requirement to further restrict the entries contained in the view. Any view may omit the filter. Although the example filters are very simple, the filter used can be as complex as necessary.

It may be desirable to limit the type of entry that the view should contain. For example, to limit this hierarchy to contain only people entries, add an ***nsfilter*** attribute to **ou=Location Views,dc=example,dc=com** with the filter value **(objectclass=organizationalperson)**.

Each view with a filter restricts the content of all descendant views, while descendant views with filters also restrict their ancestor's contents. For example, creating the top view **ou=Location Views** first together with the new filter mentioned above would create a view with all entries with the **organization** object class. When the descendant views are added that further restrict entries, the entries that now appear in the descendant views are removed from the ancestor views. This demonstrates how virtual DIT views mimic the behavior of traditional DITs.

Although virtual DIT views mimic the behavior of traditional DITs, views can do something that traditional DITs cannot: entries can appear in more than one location. For example, to associate **Entry B** with both **Mountain View** and **Sunnyvale** (see Figure 4.16, "A DIT with a Virtual DIT View Hierarchy"), add the **Sunnyvale** value to the location attribute, and the entry appears in both views.

4.4.4. Views and Other Directory Features

Both *class of service* and *roles* in Directory Server support views; see Section 4.3, "Grouping Directory Entries" . When adding a class of service or a role under a view hierarchy, the entries that are both logically and actually contained in the view are considered within scope. This means that roles and class of service can be applied using a virtual DIT view, but the effects of that application can be seen even when querying the flat namespace.

For information on using these features, see "Advanced Entry Management," in the *Red Hat Directory Server Administrator's Guide*.

The use of views requires a slightly different approach to access control. Because there is currently no explicit support for ACLs in views, create role-based ACLs at the view parent and add the roles to the appropriate parts of the view hierarchy. In this way, take advantage of the organizational property of the hierarchy.

If the base of a search is a view and the scope of the search is not a base, then the search is a views-based search. Otherwise, it is a conventional search.

For example, performing a search with a base of **dc=example,dc=com** does not return any entries from the virtual search space are returned; in fact, no virtual-search-space search is performed. Views processing occurs only if the search base is **ou=Location Views**. This way, views ensure that the search does not result in entries from both locations. (If it were a conventional DIT, entries from both locations are returned.)

4.4.5. Effects of Virtual Views on Performance

The performance of views-based hierarchies depends on the construction of the hierarchy itself and the number of entries in the DIT. In general, there may be a marginal change in performance (within a few percentage points of equivalent searches on a conventional DIT) if virtual DIT views are enabled in the directory service. If a search does not invoke a view, then there is no performance impact. Test the virtual DIT views against expected search patterns and loads before deployment.

We also recommend that the attributes used in view filters be indexed if the views are to be used as general-purpose navigation tools in the organization. Further, when a sub-filter used by views matches a configured virtual list view index, that index is used in views evaluation.

There is no need to tune any other part of the directory specifically for views.

4.4.6. Compatibility with Existing Applications

Virtual DIT views are designed to mimic conventional DITs to a high degree. The existence of views should be transparent to most applications; there should be no indication that they are working with views. Except for a few specialized cases, there is no need for directory users to know that views are being used in a Directory Server instance; views appear and behave like conventional DITs.

Certain types of applications may have problems working with a views-enabled directory service. For example:

- Applications that use the DN of a target entry to navigate up the DIT.

  This type of application would find that it is navigating up the hierarchy in which the entry physically exists instead of the view hierarchy in which the entry was found. The reason for this is that views make no attempt to disguise the true location of an entry by changing the DN of the entry to conform to the view's hierarchy. This is by design - many applications would not function if the true location of an entry were disguised, such as those applications that rely on the DN to identify a unique entry. This upward navigation by deconstructing a DN is an unusual technique for a client application, but, nonetheless, those clients that do this may not function as intended.

- Applications that use the ***numSubordinates*** operational attribute to determine how many entries exist beneath a node.

  For the nodes in a view, this is currently a count of only those entries that exist in the real search space, ignoring the virtual search space. Consequently, applications may not evaluate the view with a search.

4.5. DIRECTORY TREE DESIGN EXAMPLES

The following sections provide examples of directory trees designed to support a flat hierarchy as well as several examples of more complex hierarchies.

4.5.1. Directory Tree for an International Enterprise

To support an international enterprise, use the Internet domain name as the root point for the directory tree, then branch the tree immediately below that root point for each country where the enterprise has operations. Avoid using a country designator as the root point for the directory tree, as mentioned in Section 4.2.1.1, "Suffix Naming Conventions", especially if the enterprise is international.

Because LDAP places no restrictions on the order of the attributes in the DNs, the ***c*** attribute can represent each country branch:

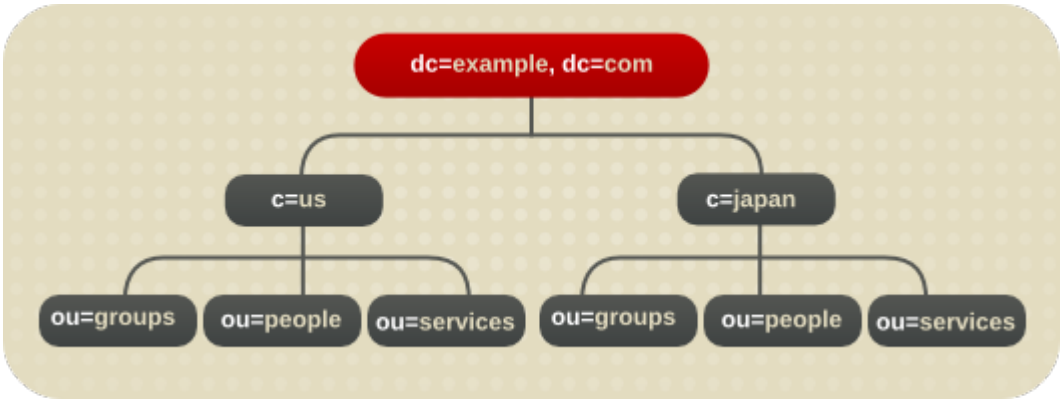


Figure 4.17. Using the c Attribute to Represent Different Countries

However, some administrators feel that this is stylistically awkward, so instead use the ***l*** attribute to represent different countries:

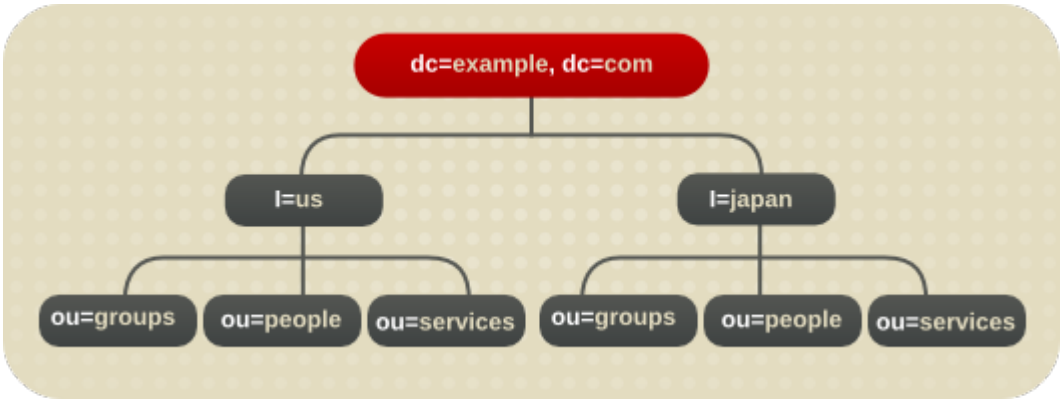


Figure 4.18. Using the l Attribute to Represent Different Countries

4.5.2. Directory Tree for an ISP

Internet service providers (ISPs) may support multiple enterprises with their directories. ISP should consider each of the customers as a unique enterprise and design their directory trees accordingly. For security reasons, each account should be provided a unique directory tree with a unique suffix and an independent security policy.

An ISP should consider assigning each customer a separate database and storing these databases on separate servers. Placing each directory tree in its own database allows data to be backed up and restored for each directory tree without affecting the other customers.

In addition, partitioning helps reduce performance problems caused by disk contention and reduces the number of accounts potentially affected by a disk outage.

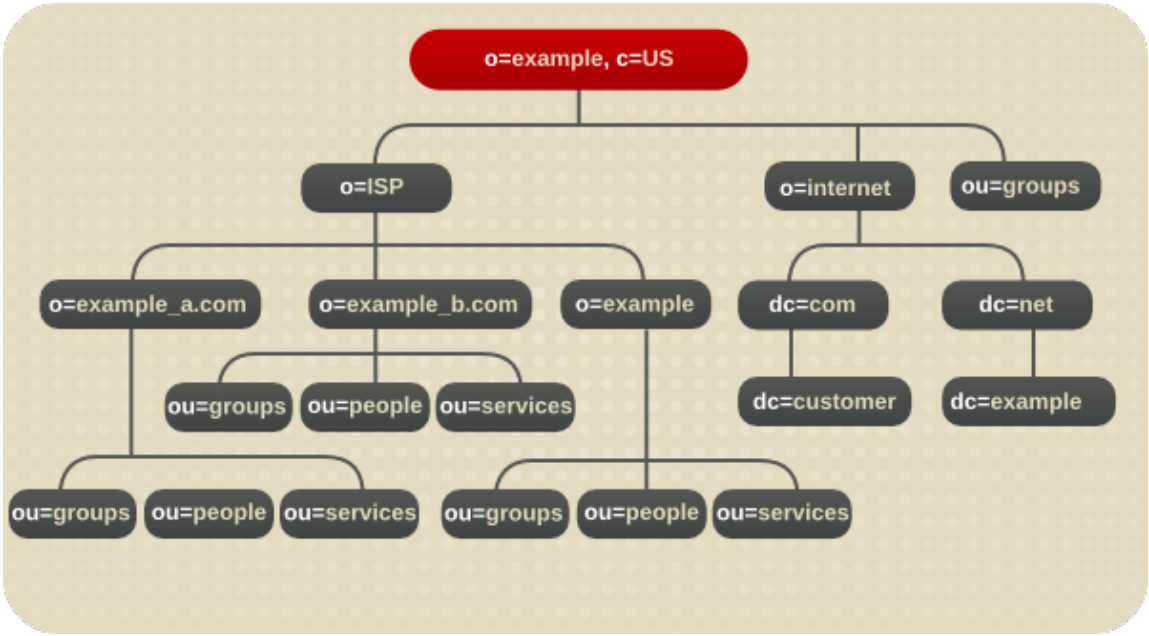


Figure 4.19. Directory tree for Example ISP

4.6. OTHER DIRECTORY TREE RESOURCES

See the following for more information about designing the directory tree:

- RFC 2247: Using Domains in LDAP/X.500 Distinguished Names
- RFC 2253: LDAPv3, UTF-8 String Representation of Distinguished Names

CHAPTER 5. DEFINING DYNAMIC ATTRIBUTE VALUES

As described in Section 3.2.2, "Standard Attributes", LDAP entries store discrete parts of information in *attributes* that are added to an entry.

Red Hat Directory Server provides several different mechanisms for dynamically and automatically maintaining some types of attributes on directory entries. These plug-ins and configuration options simplify managing directory data and expressing relationships between entries.

5.1. INTRODUCTION TO MANAGED ATTRIBUTES

When performing a site survey, one of the earliest steps is to identify the *characteristics* of the entries in the directory (Section 2.3.3, "Characterizing the Directory Data"). These characteristics are the different aspects of the entities that need to be recorded in the directory entry. For an employee, this means information like the person's manager, title, business category, email address, home and office phone numbers. Each characteristic of the entry is maintained in an entry attribute.

Part of the characteristics of entries are their *relationships* to each other. Obviously, a manager has an employee, so those two entries are related. Groups are associated with their members. There are less apparent relationships, too, like between entries which share a common physical location.

Red Hat Directory Server provides several different ways that these relationships between entries can be maintained smoothly and consistently. There are several plug-ins that can apply or generate attributes automatically as part of the data within the directory:

- *Attribute uniqueness* requires that every instance of a particular attribute within the subtree or database has a unique value. This is enforced whenever an entry is created or an attribute is modified.
- *Classes of service* use one entry as a template; whenever that attribute value changes, then all other entries within the scope of the CoS automatically have the same attribute on their entries changed. (The entries affected by the CoS are identified through a definition entry.)
- *Managed entries* create one entry according to a defined template whenever another entry in a defined scope is created. There are times, particularly with integration with external clients, when it may be necessary to have entry pairs created and managed automatically. Managed entries defines the template of the second entry and provides a mechanism to update it automatically.
- *Linked attributes* follow DN values in attributes in one entry and automatically add a pre-determined attribute (with a value that points back to the original entry) to the referenced entries. So, if entry A lists entry B as a direct report, then entry B can automatically be updated to have a ***manager*** attribute with entry A as its specified manager.
- *Distributed numeric assignments* automatically assign unique identifying numbers to entries. This is useful for GID or UID number assignments, which must be unique across an organization.

Consider several things about specific entry attribute values, as part of planning both directory data and directory schema:

- How are the entries related? Are there common attributes which are shared among entries? Are there attributes which must represent connections between entries?
- How and where (in what entry) is the original source of the data likely to be maintained? How often is this information updated and how many entries are affected when the data are changed?
- What schema elements are used by these entries and what is the syntax of those attributes?
- How does the plug-in handle distributed directory configuration, such as replication or synchronization?

5.2. ABOUT ATTRIBUTE UNIQUENESS

The Attribute Uniqueness Plug-in is a preoperation plug-in. This means that the plug-in checks all update operations *before* the server performs an LDAP operation. The plug-in determines whether the operation applies to an attribute and a suffix that it is configured to monitor.

If an update operation applies to an attribute and suffix monitored by the plug-in and it would cause two entries to have the same attribute value, then the server terminates the operation and returns an **LDAP_CONSTRAINT_VIOLATION** error to the client.

Each instance of the Attribute Uniqueness Plug-in performs a check on a single attribute for one or more subtrees. To check uniqueness of several attributes, a separate instance of the plug-in must be created for each attribute to check.

The Attribute Uniqueness Plug-in can operate in specific, user-defined ways:

- It can check every entry in the specified subtrees.

  For example, if a company, **example.com**, hosts the directories for **example_a.com** and **example_b.com**, when an entry such as **uid=jdoe,ou=people,o=example_a,dc=example,dc=com** is added, uniqueness needs to be enforced only in the **o=example_a,dc=example,dc=com** subtree. This is done by listing the DN of the subtree explicitly in the Attribute Uniqueness Plug-in configuration.

- Specify an object class pertaining to an entry in the DN of the updated entry and perform the uniqueness check on all the entries beneath it.

  This option is useful in hosted environments. For example, when adding an entry such as **uid=jdoe,ou=people,o=example_a,dc=example,dc=com**, enforce uniqueness under the **o=example_a,dc=example,dc=com** subtree without listing this subtree explicitly in the configuration but, instead, by indicating a *marker object class*. If the marker object class is set to **organization**, the uniqueness check algorithm locates the entry in the DN that has this object class (**o=example_a**) and performs the check on all entries beneath it.

  Additionally, it is possible to check uniqueness only if the updated entry includes a specified object class. For example, a check may be performed only if the updated entry includes **objectclass=inetorgperson**.

Directory Server provides a default instance of the Attribute Uniqueness Plug-in for the ***uid*** attribute when the Directory Server was first set up. This plug-in instance ensures that values given to the ***uid*** attribute are unique in the root suffix (the suffix corresponding to the **userRoot** database).

This plug-in is disabled by default because it affects the operation of multi-master replication.

Attribute Uniqueness Plug-ins do not perform any checking on attribute values when an update is performed as part of a replication operation.

Because all modifications by client applications are performed on the supplier server, the Attribute Uniqueness Plug-in should be enabled on the supplier. It is unnecessary to enable it on the consumer server.

Enabling the Attribute Uniqueness Plug-in on the consumer does not prevent Directory Server from operating correctly but is likely to cause a performance degradation.

In a multi-master replication scenario, the masters act both as suppliers and consumers of the same replica. Because multi-master replication uses a loosely consistent replication model, enabling an Attribute Uniqueness Plug-in on one of the servers is not sufficient to ensure that attribute values will be unique across both supplier servers at any given time. Therefore, enabling an Attribute Uniqueness Plug-in on one server can cause inconsistencies in the data held on each replica.

However, it is possible to use an Attribute Uniqueness Plug-in, providing both of the following conditions are met:

- The attribute on which the uniqueness check is performed is a naming attribute.
- The Attribute Uniqueness Plug-in is enabled on both supplier servers.

When these conditions are met, attribute uniqueness conflicts are reported as naming conflicts at replication time. Naming conflicts require manual resolution.

5.3. ABOUT CLASSES OF SERVICE

A *class of service* (CoS) shares attributes between entries in a way that is invisible to applications. With CoS, some attribute values may not be stored with the entry itself. Instead, they are generated by class of service logic as the entry is sent to the client application.

For example, the directory contains thousands of entries that all share the common attribute ***facsimileTelephoneNumber***. Traditionally, to change the fax number required updating each entry individually, a large job for administrators that runs the risk of not updating all entries. With CoS, the attribute value can be generated dynamically. The ***facsimileTelephoneNumber*** attribute is stored in one location, and each entry retrieves its fax number attribute from that location. For the application, these attributes appear just like all other attributes, despite not actually being stored on the entries themselves.

Each CoS is comprised of the several entries in the directory:

- The *CoS definition entry* identifies the type of CoS. It is stored as an LDAP subentry below the branch it affects.
- The *template entry* contains a list of the shared attribute values. Changes to the template entry attribute values are automatically applied to all the entries sharing the attribute.

The CoS definition entry and the template entry interact to provide attribute values to their *target entries*, the entries within their scope. The value they provide depends upon the following:

- The entry's DN (different portions of the directory tree might contain different CoS).
- A service class attribute value stored with the entry.

  The absence of a service class attribute can imply a specific default CoS.
- The attribute value stored in the CoS template entry.

  Each CoS template entry supplies the attribute value for a particular CoS.
- The object class of the entry.

  CoS attribute values are generated only when an entry contains an object class allowing the attribute when schema checking is turned on; otherwise, all attribute values are generated.
- The attribute stored in some particular entry in the directory tree.

NOTE

Roles and the 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. For example, use a role-based attribute to set the server look-through limit on a role-by-role basis.

5.3.1. About a Pointer CoS

A pointer CoS identifies the template entry using the template DN only. There may be only one template DN for each pointer CoS. A pointer CoS applies to all entries within the scope of the template entry.

For example, the pointer CoS in Figure 5.1, "Sample Pointer CoS" shares a common postal code with all of the entries stored under **dc=example,dc=com**.

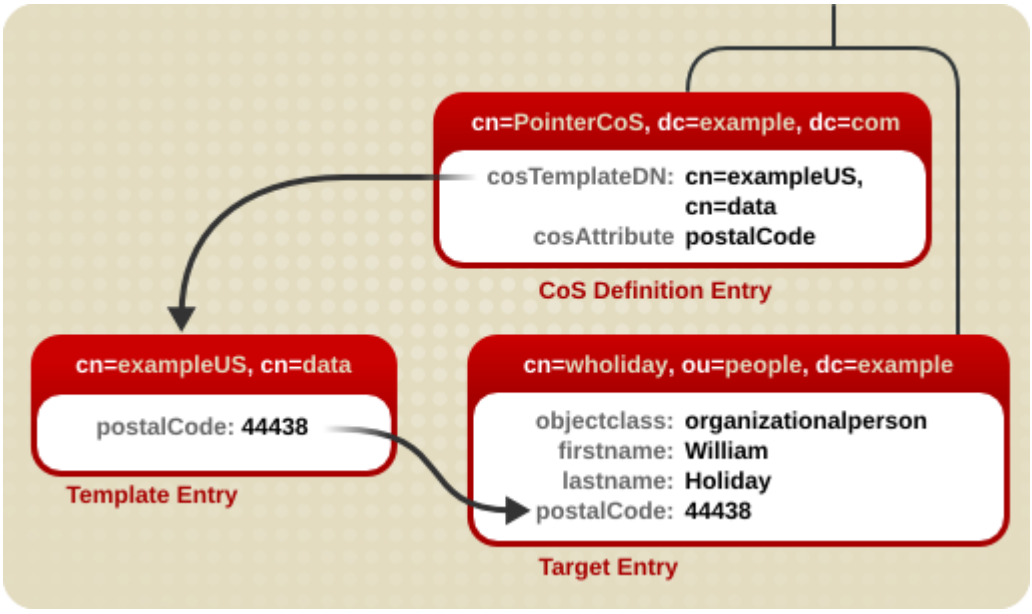


Figure 5.1. Sample Pointer CoS

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 the entry **cn=wholiday,ou=people,dc=example,dc=com**, the Directory Server returns the value available in the template entry **cn=exampleUS,cn=data**.

5.3.2. About an Indirect CoS

An indirect CoS identifies the template entry using the value of one of the target entry's attributes. The target entry's attribute must contain the DN of an existing entry.

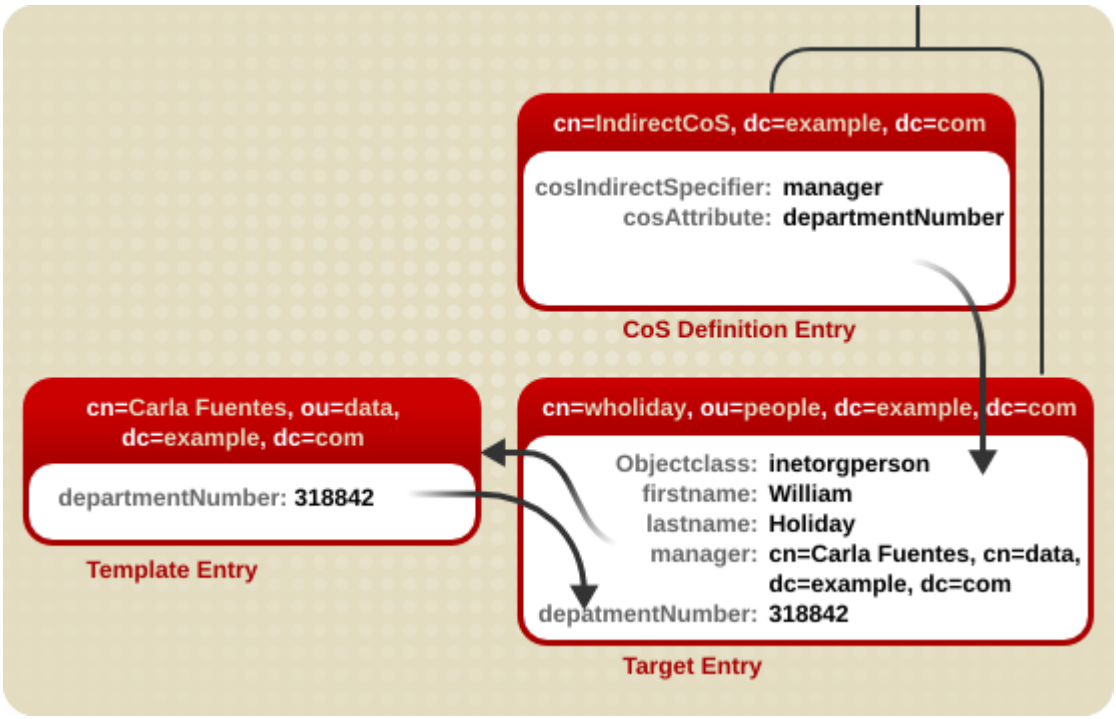


Figure 5.2. Sample Indirect CoS

In this example, the target entry for William Holiday contains the indirect specifier, the ***manager*** attribute. William's manager is Carla Fuentes, so the ***manager*** attribute contains a pointer to the DN of the template entry, **cn=Carla Fuentes,ou=people,dc=example,dc=com**. The template entry in turn provides the ***departmentNumber*** attribute value of **318842**.

5.3.3. About a Classic CoS

A classic CoS identifies the *template entry* by both its DN and the value of one of the target entry's attributes. Classic CoS can have multiple template entries, including a default CoS template to be applied to those entries that do not belong to any other CoS template.

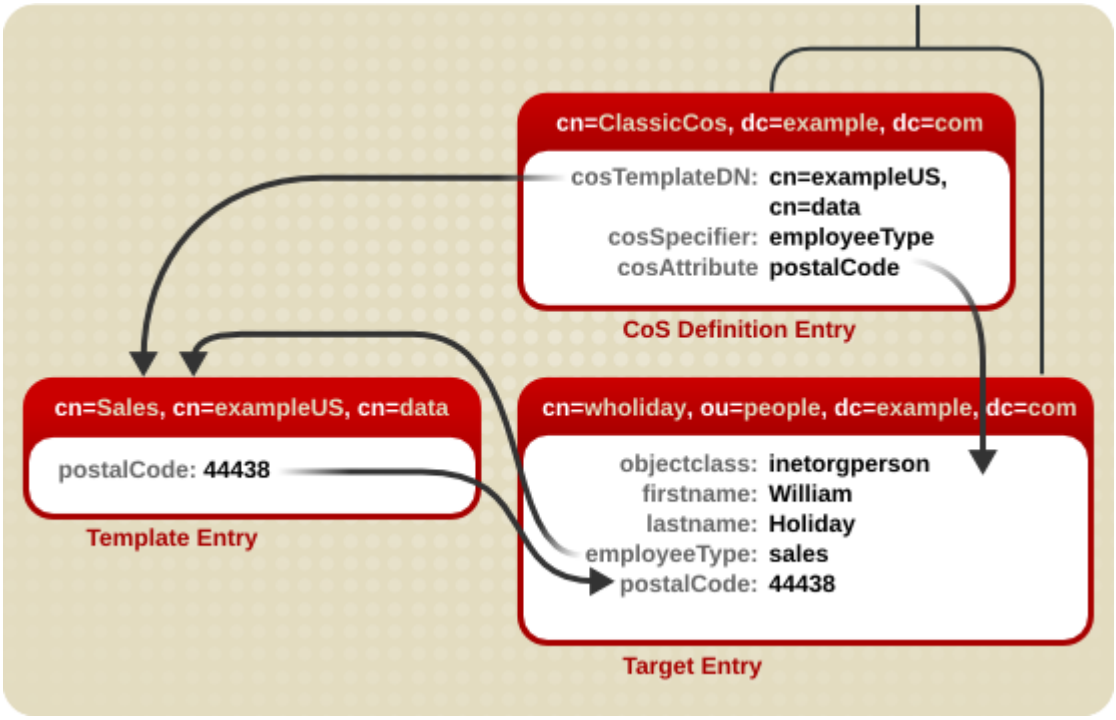


Figure 5.3. Sample Classic CoS

In this example, the CoS definition entry's ***cosSpecifier*** attribute specifies the ***employeeType*** attribute. This attribute, in combination with the template DN, identify 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.

5.4. ABOUT MANAGED ENTRIES

Some clients and integration with Red Hat Directory Server require dual entries. For example, Posix systems typically have a group for each user. The Directory Server's Managed Entries Plug-in creates a new managed entry, with accurate and specific values for attributes, automatically whenever an appropriate origin entry is created.

The basic idea is that there are situations when Entry A is created and there should automatically be an Entry B with related attribute values. For example, when a Posix user (**posixAccount** entry) is created, a corresponding group entry (**posixGroup** entry) should also be created. An instance of the Managed Entries Plug-in identifies what entry (the *origin entry*) triggers the plug-in to automatically generate a new entry (the *managed entry*). It also identifies a separate template entry which defines the managed entry configuration.

The instance of the Managed Entries Plug-in defines three things:

- The search criteria to identify the origin entries (using a search scope and a search filter)
- The subtree under which to create the managed entries (the new entry location)
- The template entry to use for the managed entries

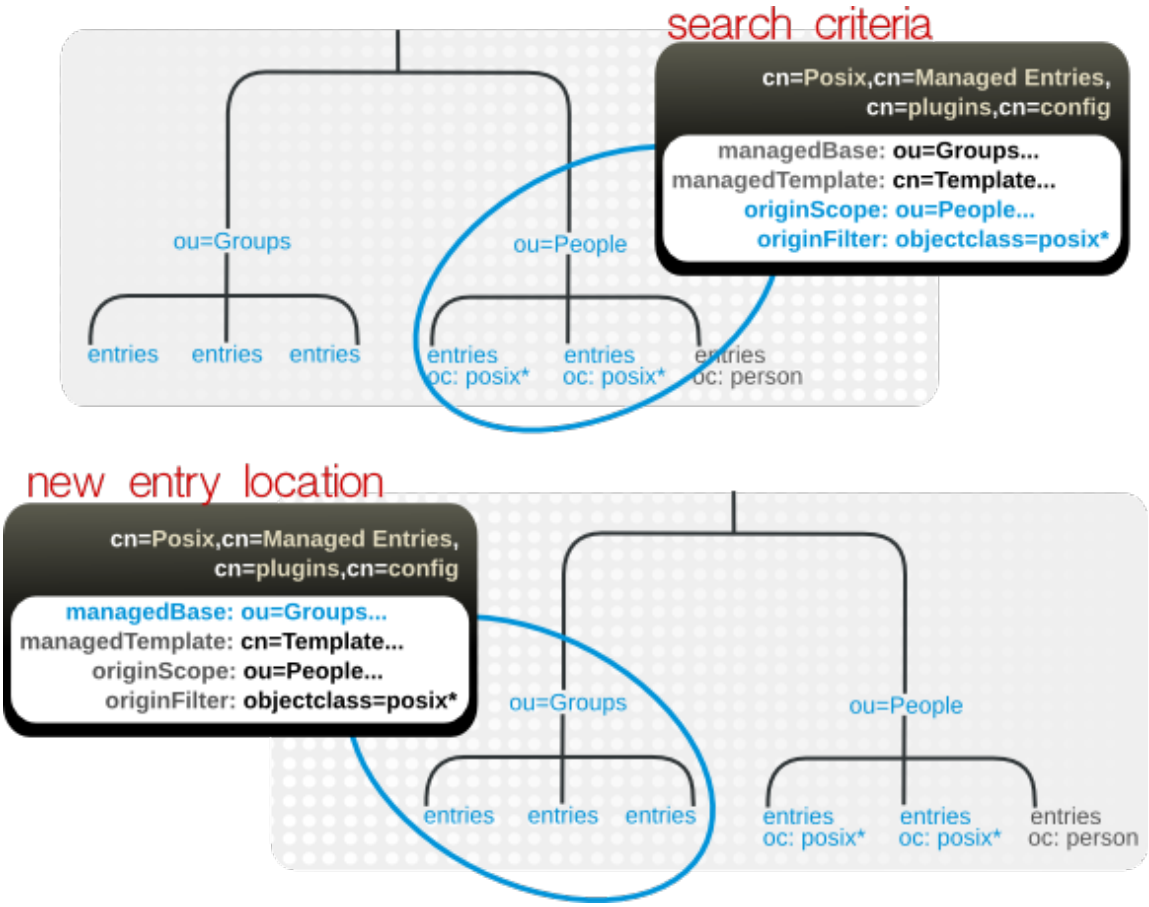


Figure 5.4. Defining Managed Entries

For example:

```
dn: cn=Posix User-Group,cn=Managed Entries,cn=plugins,cn=config
objectclass: extensibleObject
cn: Posix User-Group
originScope: ou=people,dc=example,dc=com
originFilter: objectclass=posixAccount
managedBase: ou=groups,dc=example,dc=com
managedTemplate: cn=Posix User-Group Template,ou=Templates,dc=example,dc=com
```

The origin entry does not have to have any special configuration or settings to create a managed entry; it simply has to be created within the scope of the plug-in and match the given search filter.

5.4.1. Defining the Template for Managed Entries

The template entry lays out the entire configuration of the managed entry, using static attributes (ones with pre-defined values) and mapped attributes (mapped attributes that pull their values from the origin entry).

```
dn: cn=Posix User-Group Template,ou=Templates,dc=example,dc=com
objectclass: mepTemplateEntry
cn: Posix User-Group Template
mepRDNAttr: cn
mepStaticAttr: objectclass: posixGroup
mepMappedAttr: cn: $cn Group Entry
mepMappedAttr: gidNumber: $gidNumber
mepMappedAttr: memberUid: $uid
```

The mapped attributes in the template use tokens, prepended by a dollar sign ($), to pull in values from the origin entry and use it in the managed entry.

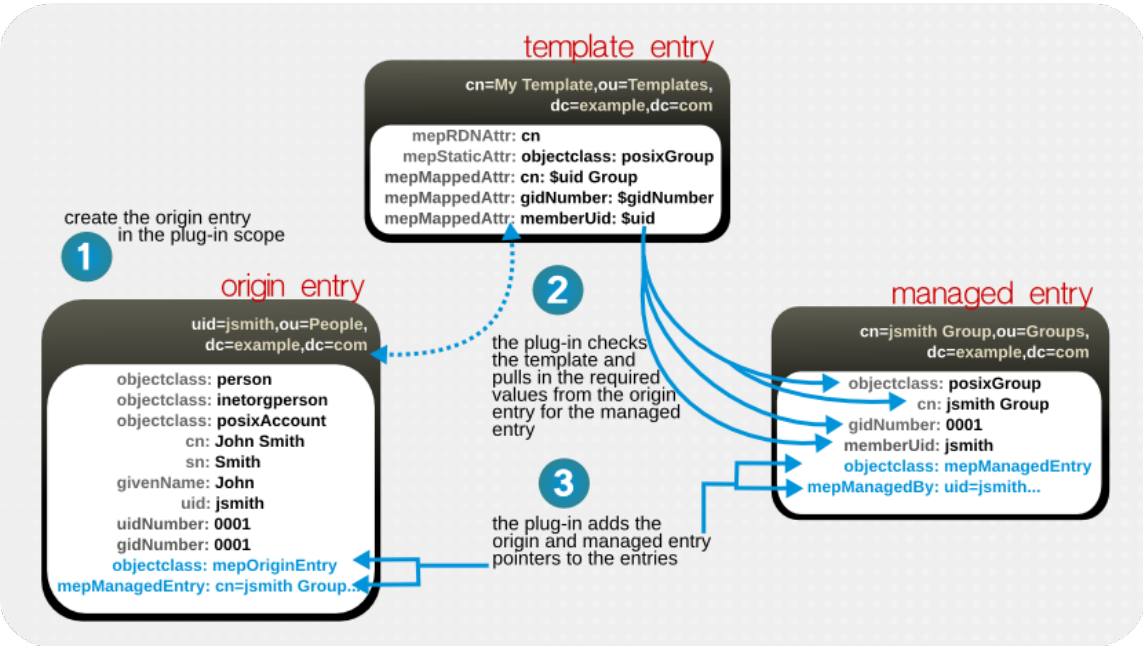


Figure 5.5. Managed Entries, Templates, and Origin Entries

NOTE

Make sure that the values given for static and mapped attributes comply with the required attribute syntax.

5.4.2. Entry Attributes Written by the Managed Entries Plug-in

Both the origin entry and the managed entry have special managed entries attributes which indicate that they are being managed by an instance of the Managed Entries Plug-in. For the origin entry, the plug-in adds links to associated managed entries.

```
dn: uid=jsmith,ou=people,dc=example,dc=com
objectclass: mepOriginEntry
objectclass: posixAccount
...
sn: Smith
mail: jsmith@example.com
mepManagedEntry: cn=jsmith Posix Group,ou=groups,dc=example,dc=com
```

On the managed entry, the plug-in adds attributes that point back to the origin entry, in addition to the attributes defined in the template.

```
dn: cn=jsmith Posix Group,ou=groups,dc=example,dc=com
objectclass: mepManagedEntry
objectclass: posixGroup
...
mepManagedBy: uid=jsmith,ou=people,dc=example,dc=com
```

Using special attributes to indicate managed and origin entries makes it easy to identify the related entries and to assess the changes made by the Managed Entries Plug-in.

5.4.3. Managed Entries Plug-in and Directory Server Operations

The Managed Entries Plug-in has some impact on how the Directory Server carries out common operations, like add and delete operations:

- *Add*. With every add operation, the server checks to see if the new entry is within the scope of any Managed Entries Plug-in instance. If it meets the criteria for an origin entry, then a managed entry is created and managed entry-related attributes are added to both the origin and managed entry.

- *Modify*. If an origin entry is modified, it triggers the plug-in to update the managed entry.

  Changing a *template* entry, however, does not update the managed entry automatically. Any changes to the template entry are not reflected in the managed entry until after the next time the origin entry is modified.

  The mapped managed attributes *within* a managed entry cannot be modified manually, only by the Managed Entry Plug-in. Other attributes in the managed entry (including static attributes added by the Managed Entry Plug-in) can be modified manually.

- *Delete*. If an origin entry is deleted, then the Managed Entries Plug-in will also delete any managed entry associated with that entry.

  There are some limits on what entries can be deleted.

  - A template entry cannot be deleted if it is currently referenced by a plug-in instance definition.

  - A managed entry cannot be deleted except by the Managed Entries Plug-in.

- *Rename*. If an origin entry is renamed, then plug-in updates the corresponding managed entry. If the entry is moved *out* of the plug-in scope, then the managed entry is deleted, while if an entry is moved *into* the plug-in scope, it is treated like an add operation and a new managed entry is created.

  As with delete operations, there are limits on what entries can be renamed or moved.

  - A configuration definition entry cannot be moved out of the Managed Entries Plug-in container entry. If the entry is removed, that plug-in instance is inactivated.

  - If an entry is moved *into* the Managed Entries Plug-in container entry, then it is validated and treated as an active configuration definition.

  - A template entry cannot be renamed or moved if it is currently referenced by a plug-in instance definition.

  - A managed entry cannot be renamed or moved except by the Managed Entries Plug-in.

- *Replication*. The Managed Entries Plug-in operations *are not initiated by replication updates*. If an add or modify operation for an entry in the plug-in scope is replicated to another replica, that operation does not trigger the Managed Entries Plug-in instance on the replica to create or update an entry. The only way for updates for managed entries to be replicated is to replicate the final managed entry over to the replica.

5.5. ABOUT LINKING ATTRIBUTES

A class of service dynamically supplies attribute values for entries which all have attributes with the *same value*, like building addresses, postal codes, or main office numbers. These are shared attribute values, which are updated in a single template entry.

Frequently, though, there are relationships between entries where there needs to be a way to express linkage between them, but the values (and possibly even the attributes) that express that relationship are different. Red Hat Directory Server provides a way to link specified attributes together, so that when one attribute in one entry is altered, a corresponding attribute on a related entry is automatically updated. The first attribute has a DN value, which points to the entry to update; the second entry attribute also has a DN value which is a back-pointer to the first entry.

For example, group entries list their members in attributes like ***member***. It is natural for there to be an indication in the user entry of what groups the user belongs to; this is set in the ***memberOf*** attribute. The ***memberOf*** attribute is a *managed* attribute through the MemberOf Plug-in. The plug-in polls every group entry for changes in their respective member attributes. Whenever a group member is added or dropped from the group, the corresponding user entry is updated with changed ***memberOf*** attributes. In this way, the ***member*** (and other member attributes) and ***memberOf*** attributes are *linked*.

The MemberOf Plug-in is limited to a single instance, exclusively for a (single) group member attribute (with some other behaviors unique to groups, like handling nested groups). Another plug-in, the Linked Attributes Plug-in, allows multiple instances of the plug-in. Each instance configures one attribute which is manually maintained by the administrator (***linkType***) and one attribute which is automatically maintained by the plug-in (***managedType***).

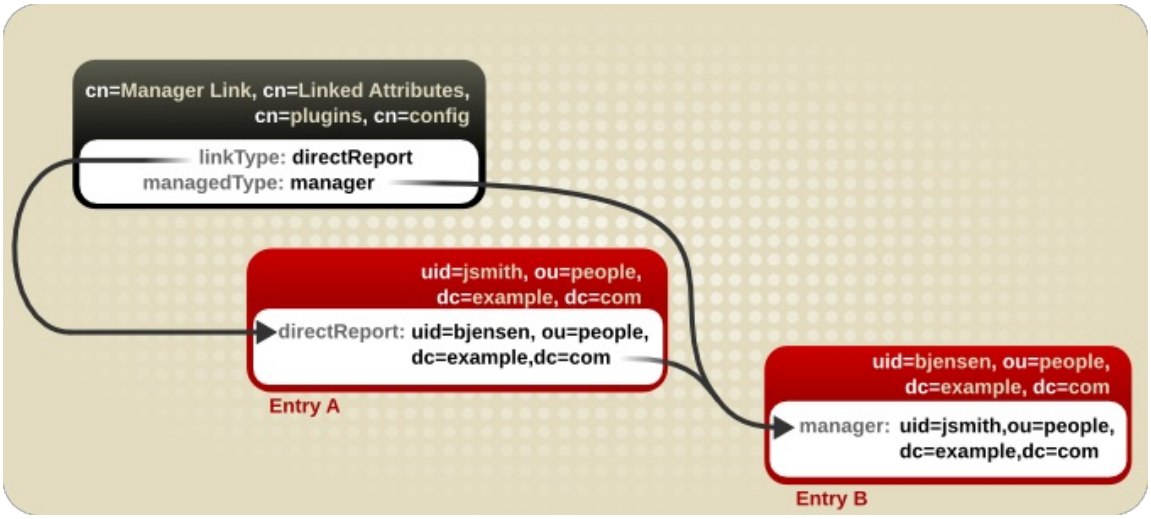


Figure 5.6. Basic Linked Attribute Configuration

NOTE

To preserve data consistency, only the plug-in process should maintain the managed attribute. Consider creating an ACI that will restrict all write access to any managed attribute.

A Linked Attribute Plug-in instance can be restricted to a single subtree within the directory. This can allow more flexible customization of attribute combinations and affected entries. If no scope is set, then the plug-in operates on the entire directory.

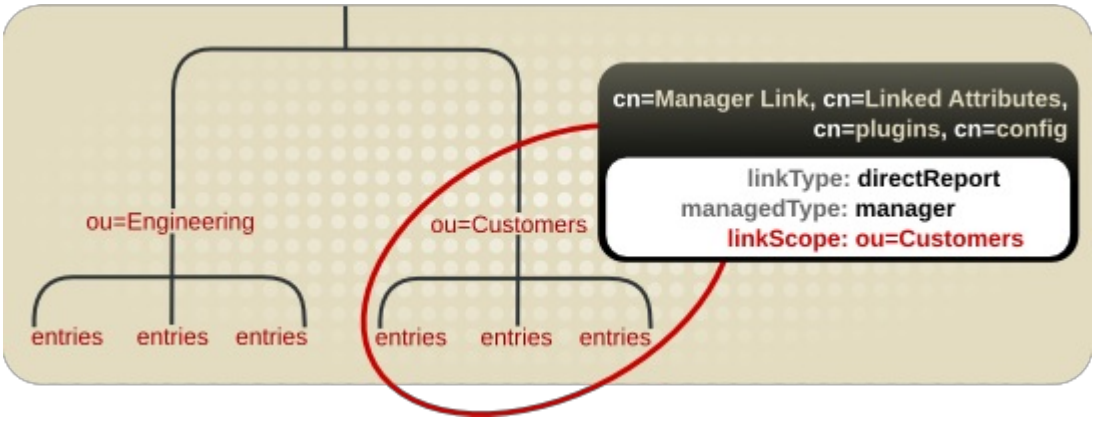


Figure 5.7. Restricting the Linked Attribute Plug-in to a Specific Subtree

5.5.1. Schema Requirements for Linking Attributes

Both the managed attribute and linked attribute must require the Distinguished Name syntax in their attribute definitions. The plug-in identifies the entries to maintain by pulling the DN from the link attribute, and then it automatically assigns the original entry DN as the value of the managed attribute. This means that both attributes have to use DNs as their values.

The managed attribute must be multi-valued. Users may be members of more than one group, be authors of more than one document, or have multiple "see also" reference entries. If the managed attribute is single-valued, then the value is not updated properly. This is not much of an issue with the default schema because many of the standard elements are multi-valued. It is particularly important to consider, however, when using custom schema.

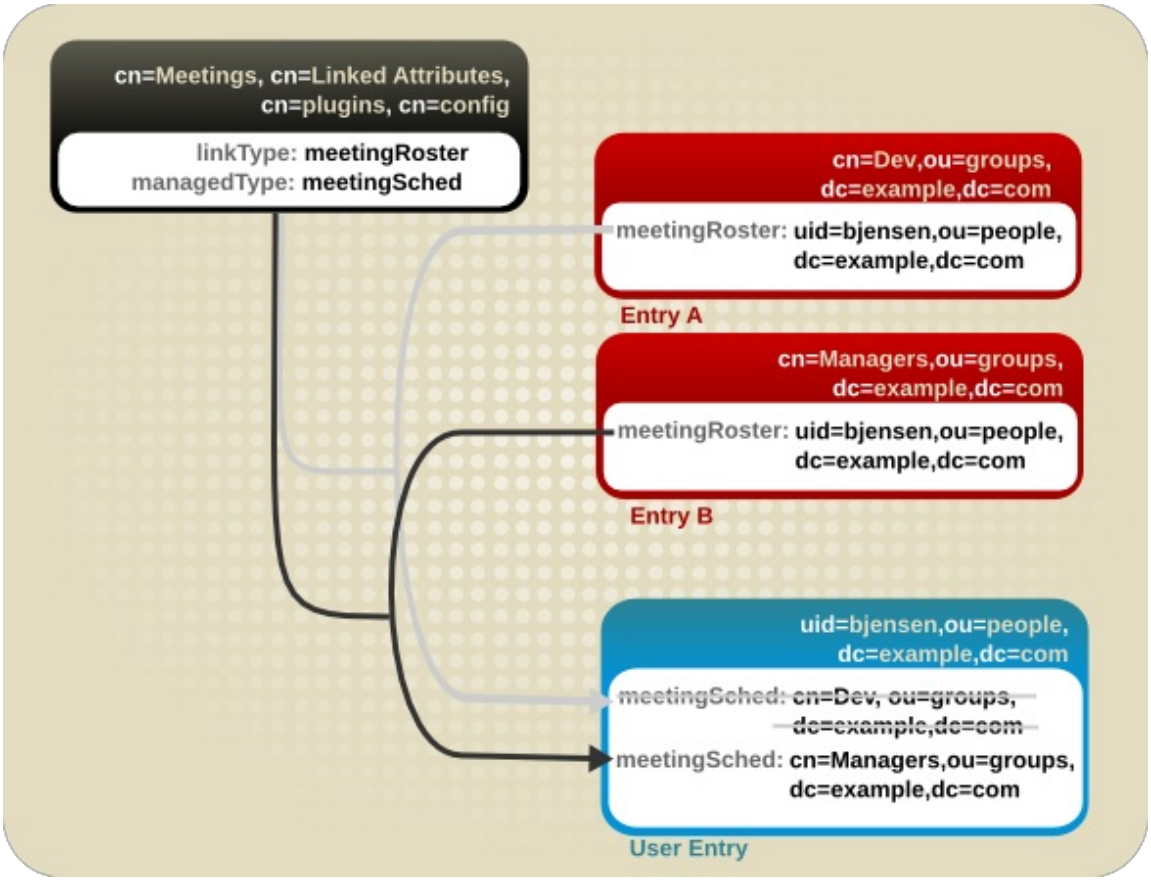


Figure 5.8. Wrong: Using a Single-Valued Linked Attribute

5.5.2. Using Linked Attributes with Replication

In a simple replication scenario (supplier-consumer), then the plug-in only has to exist on the supplier, because no writes can be made on the consumer.

For multi-master replication, each supplier must have its own plug-in instance, all configured identically, and the managed attributes must be excluded from replication using fractional replication.

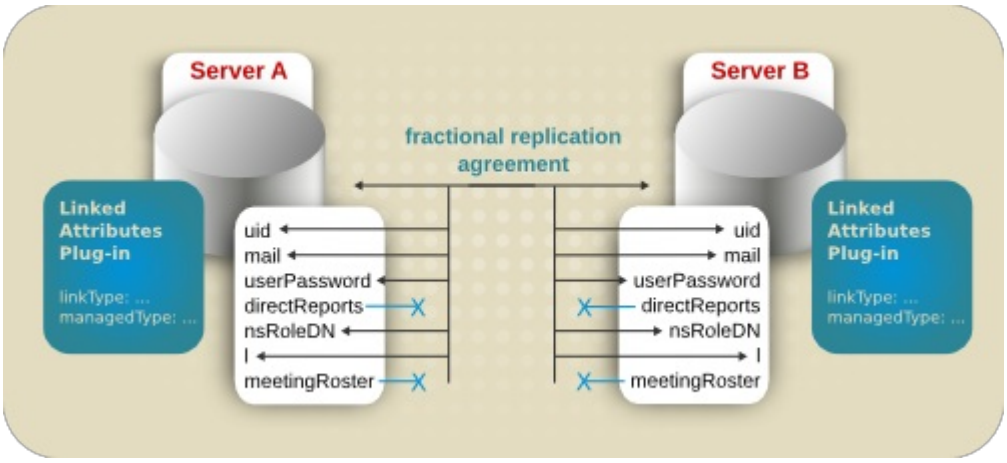


Figure 5.9. Linking Attributes and Replication

Any changes that are made on one supplier automatically trigger the plug-in to manage the values on the corresponding directory entries, so the data stay consistent across servers. However, the managed attributes must be maintained by the plug-in instance for the data to be consistent between the linked entries. This means that managed attribute values should be maintained solely by the plug-in processes, not the replication process, even in a multi-master replication environment.

5.6. ABOUT DYNAMICALLY ASSIGNING UNIQUE NUMBER VALUES

Some entry attributes require having a unique number, such as ***uidNumber*** and ***gidNumber***. The Directory Server can automatically generate and supply unique numbers for specified attributes using the Distributed Numeric Assignment (DNA) Plug-in.

Many situations require a unique numeric attribute, such as UID/GID numbers or PIN numbers. The server uses DNA Plug-in instances to specify the attributes for which to generate the numbers, so whenever that attribute is added to an entry, the unique value can be assigned.

NOTE

Attribute uniqueness is not necessarily preserved with the DNA Plug-in. The plug-in only assigns non-overlapping ranges, but it does allow manually-assigned numbers for its managed attributes, and it does not verify or require that the manually-assigned numbers are unique.

5.6.1. How the Directory Server Manages Unique Numbers

The issue with assigning unique numbers is not with generating the numbers but in effectively managing numbers so that there are not any conflicts with other assigned numbers when entries are replicated and that every server has a sufficient range of numbers to assign.

The DNA Plug-in for a server assigns a range of available numbers that that instance can issue. The range definition is very simple and is set by two attributes: the server's next available number (the low end of the range) and its maximum value (the top end of the range). The initial bottom range is set when the plug-in instance is configured. After that, the bottom value is updated by the plug-in. By breaking the available numbers into ranges, the servers can all continually assign numbers without overlapping with each other.

The server performs a sorted search, internally, to see if the next specified range is already taken, requiring the managed attribute to have an equality index with the proper ordering matching rule.

For multi-master replication, each supplier can be configured with a threshold, so that when it begins running out of numbers in its range, it can request additional ranges from other suppliers. Each supplier keeps a track of its current range in a separate configuration entry. The configuration entry is replicated to all of the other suppliers, so each supplier can check that configuration to find a server to contact for a new range.

The range set on the individual servers and the range configuration entries are how the Directory Server *distributes* numbers efficiently for entries.

The DNA Plug-in can assign unique numbers to a single attribute type or across multiple attribute types from a single range of unique numbers.

This provides several options for assigning unique numbers to attributes:

- A single number assigned to a single attribute type from a single range of unique numbers.
- The same unique number assigned to two attributes for a single entry.
- Two different attributes assigned two different numbers from the same range of unique numbers.

In many cases, it is sufficient to have a unique number assigned per attribute type. When assigning an ***employeeID*** to a new employee entry, it is important each employee entry is assigned a unique ***employeeID***.

However, there are cases where it may be useful to assign unique numbers from the same range of numbers to multiple attributes. For example, when assigning a ***uidNumber*** and a ***gidNumber*** to a **posixAccount** entry, the DNA Plug-in can be configured to assign the same number to both attributes.

The DNA Plug-in is applied, always, to a specific area of the directory tree (the *scope*) and to specific entry types within that subtree (the *filter*).

Frequently, entirely different users are stored in different branches of the directory tree. For example, a hosting service may have one clients' users in the **ou=Example Corp.** branch and another clients' users in the **ou=Acme Company** branch. In this case, the assigned numbers have to be unique *within the subtree* but not necessarily across the entire directory. In this case, it is all right for Barbara Jensen in the **ou=Example Corp.** branch to have **uidNumber:5** in her entry *and* for John Smith in the **ou=Acme Company** branch to have **uidNumber:5** in his entry, because these are separate organizations. Applying ranges to a specific subtree is set in the DNA scope, such as **dnaScope: ou=people,dc=example,dc=com**.

The unique number can also be distinguished between ranges by using a prefix to identify the different kinds of user entries. For example, if the DNA prefix is set to **acme**, then the unique numbers in the Acme Company branch have **acme** in front of the number, like **uid: acme5**.

5.6.2. Using DNA to Assign Values to Attributes

There are several different ways that the Directory Server can handle generating attribute values.

In the simplest case, a user entry is added to the directory with an object class which requires the unique-number attribute, but without the attribute. Adding (or requiring) the managed attribute without a value triggers the DNA Plug-in to assign a value. When an entry is added, the plug-in checks whether the entry matches the defined range according to the scope and filter set for the plug-in. If the entry matches the range and the attribute that DNA is managing for that range is missing from the entry being added, then the DNA Plug-in assigns the next value. This option only works if the DNA Plug-in has been configured to assign unique values to a single attribute.

For example, the **posixAccount** object class requires the ***uidNumber*** attribute. If the ***uidNumber*** attribute is managed by the DNA Plug-in and a user entry is added without the ***uidNumber*** attribute within the scope of the filter, then the server checks the new entry, sees that it needs the managed ***uidNumber*** attribute, and adds the attribute with an automatically assigned value.

```
ldapmodify -a -D "cn=directory manager" -W -p 389 -h server.example.com -x

dn: uid=jsmith,ou=people,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: posixAccount
uid: jsmith
cn: John Smith
....
```

The plug-in processes the missing attribute, asks for the next available number from the server, and supplies the value for the entry.

A similar and more manageable option is to use a *magic number*. This magic number is a template value for the managed attribute, something outside the server's range, a number or even a word, that the plug-in recognizes it needs to replace with a new assigned value. When an entry is added with that number, and the entry is within the scope and filter of the configured DNA Plug-in, then using the magic number automatically triggers the plug-in to generate a new value.

When the DNA Plug-in has been configured to assign the same unique number to both a ***uidNumber*** and a ***gidNumber*** to a **posixAccount** entry, the DNA Plug-in will assign the same number to both attributes. To do this, then pass both managed attributes to the modify operation, specifying the magic number. For example:

```
ldapmodify -a -D "cn=directory manager" -W -p 389 -h server.example.com -x

dn: uid=jsmith,ou=people,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: posixAccount
uid: jsmith
cn: John Smith
uidNumber: magic
gidNumber: magic
....
```

The magic number is very useful for importing entries from LDIF or for triggering the DNA Plug-in to generate unique numbers for several different attributes.

The DNA Plug-in only generates new, unique values. If an entry is added or modified to use a specific value for an attribute controlled by the DNA Plug-in, the specified number is used; the DNA Plug-in will not overwrite it.

NOTE

Attribute uniqueness is not necessarily preserved with the DNA Plug-in. The plug-in only assigns non-overlapping ranges, but it does allow manually-assigned numbers for its managed attributes, and it does not verify or require that the manually-assigned numbers are unique.

5.6.3. Using the DNA Plug-in with Replication

With multi-master replication, there are two entries referenced by the server:

- The managed ranges for the DNA Plug-in
- A shared configuration entry which stores the information about the server's available ranges

When the plug-in instance is created, then the DNA Plug-in automatically creates an entry beneath the shared configuration entry with the supplier configuration. For example:

```
dn: dnaHostname=ldap1.example.com+dnaPortNum=389,cn=Account UIDs,ou=Ranges,dc=example,dc=com
objectClass: extensibleObject
objectClass: top
dnahostname: ldap1.example.com
dnaPortNum: 389
dnaSecurePortNum: 636
dnaRemainingValues: 1000
```

When a server needs a new range of numbers, it searches the configuration entries under the container entry. When it finds the server with the highest available range, it sends an extended operation request to have part of the range assigned to it. If the second server agrees, the second server sends the requesting server the new range assignment.

CHAPTER 6. DESIGNING THE DIRECTORY TOPOLOGY

Chapter 4, *Designing the Directory Tree* covers how to design the directory service stores entries. Because Red Hat Directory Server can store a large number of entries, it is possible to distribute directory entries across more than one server. The directory's topology describes how the directory tree is divided among multiple physical Directory Servers and how these servers link with one another.

This chapter describes planning the topology of the directory service.

6.1. TOPOLOGY OVERVIEW

Directory Server can support a *distributed directory*, where the directory tree (designed in Chapter 4, *Designing the Directory Tree*) is spread across multiple physical Directory Servers. The way the directory is divided across those servers helps accomplish the following:

- Achieve the best possible performance for directory-enabled applications.
- Increase the availability of the directory service.
- Improve the management of the directory service.

The database is the basic unit for jobs such as replication, performing backups, and restoring data. A single directory can be divided into manageable pieces and assigned to separate databases. These databases can then be distributed between a number of servers, reducing the workload for each server. More than one database can be located on a single server. For example, one server might contain three different databases.

When the directory tree is divided across several databases, each database contains a portion of the directory tree, called a *suffix*. For example, one database can be used to store only entries in the **ou=people,dc=example,dc=com** suffix, or branch, of the directory tree.

When the directory is divided between several servers, each server is responsible for only a part of the directory tree. The distributed directory service works similarly to the Domain Name Service (DNS), which assigns each portion of the DNS namespace to a particular DNS server. Likewise, the directory namespace can be distributed across servers while maintaining a directory service that, from a client's point of view, appears to be a single directory tree.

The Directory Server also provides *knowledge references*, mechanisms for linking directory data stored in different databases. Directory Server includes two types of knowledge references; *referrals* and *chaining*.

The remainder of this chapter describes databases and knowledge references, explains the differences between the two types of knowledge references, and describes how to design indexes to improve the performance of the databases.

6.2. DISTRIBUTING THE DIRECTORY DATA

Distributing the data allows the directory service to be scaled across multiple servers without physically containing those directory entries on each server in the enterprise. A distributed directory can therefore hold a much larger number of entries than would be possible with a single server.

In addition, the directory service can be configured to hide the distribution details from the user. As far as users and applications are concerned, there is only a single directory that answers their directory queries.

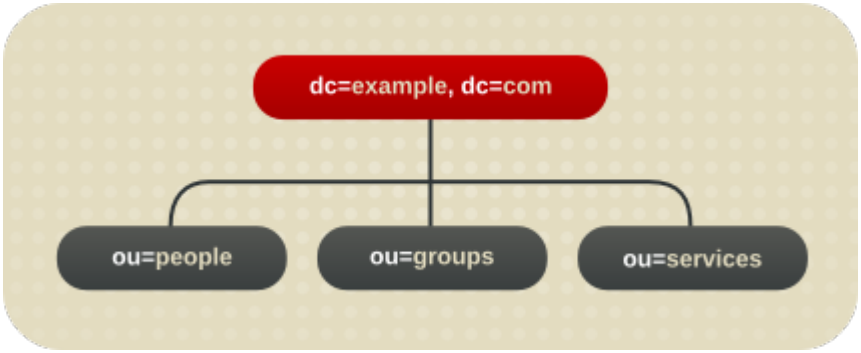
The following sections describe the mechanics of data distribution in more detail:

- Section 6.2.1, “About Using Multiple Databases”
- Section 6.2.2, “About Suffixes”

6.2.1. About Using Multiple Databases

Directory Server stores data in LDBM databases. This a high-performance, disk-based database. Each database consists of a set of large files that contain all of the data assigned to it.

Different portions of the directory tree can be stored in different databases.



For example, Figure 6.1, "Storing Suffix Data in Separate Databases" shows three suffixes being stored in three separate databases.

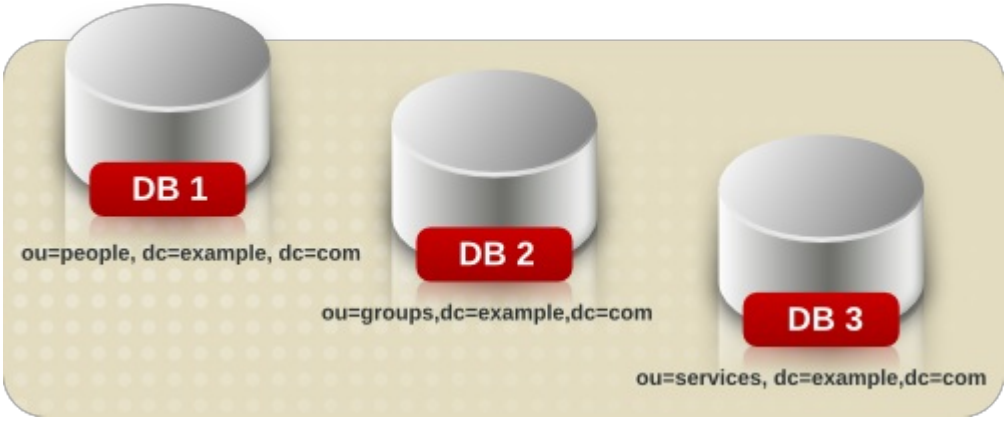


Figure 6.1. Storing Suffix Data in Separate Databases

When the directory tree is divided between a number of databases, these databases can then be distributed across multiple servers. For example, if there are three databases, DB1, DB2, and DB3, to contain the three suffixes of the directory tree, they can be stored on two servers, Server A and Server B.

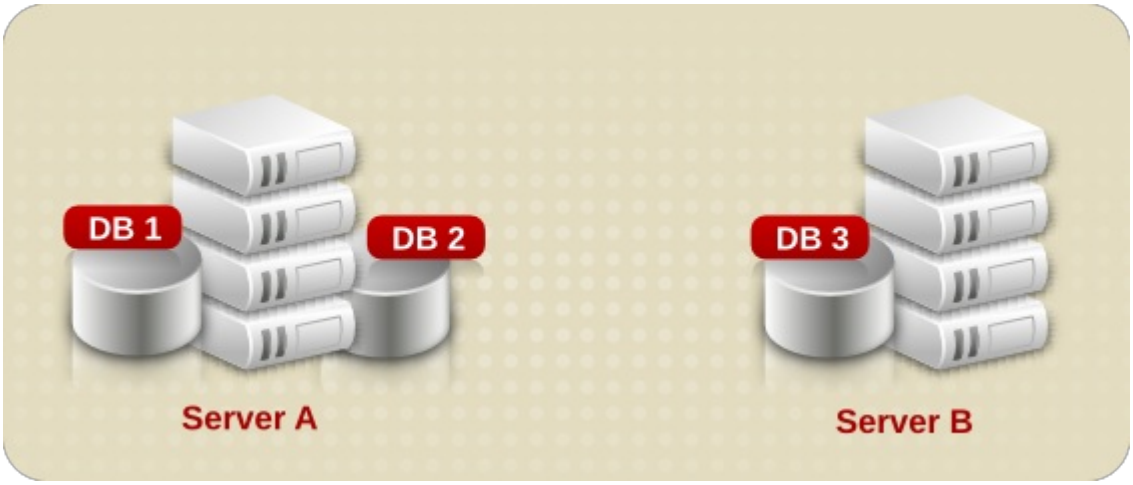


Figure 6.2. Dividing Suffix Databases Between Separate Servers

Server A contains DB1 and DB2, and Server B contains DB3.

Distributing databases across multiple servers reduces the workload on each server. The directory service can therefore be scaled to a much larger number of entries than would be possible with a single server.

In addition, Directory Server supports adding databases dynamically, which means that new databases can be added when the directory service needs them without taking the entire directory service off-line.

6.2.2. About Suffixes

Each database contains the data within a specific suffix of the Directory Server. Both root and subsuffixes can be created to organize the contents of the directory tree. A root suffix is the entry at the top of a tree. It can be the root of the directory tree or part of a larger tree designed for the Directory Server. A subsuffix is a branch beneath a root

suffix. The data for root and subsuffixes are contained by databases.

For example, Example Corp. creates suffixes to represent the distribution of their directory data.

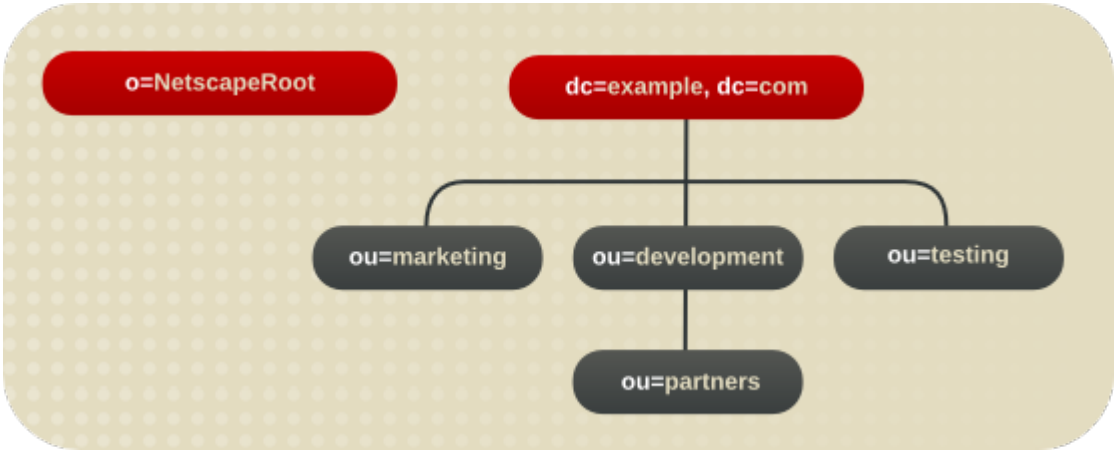


Figure 6.3. Directory Tree for Example Corp.

Example Corp. can spread their directory tree across five different databases, as in Figure 6.4, "Directory Tree Spread across Multiple Databases".

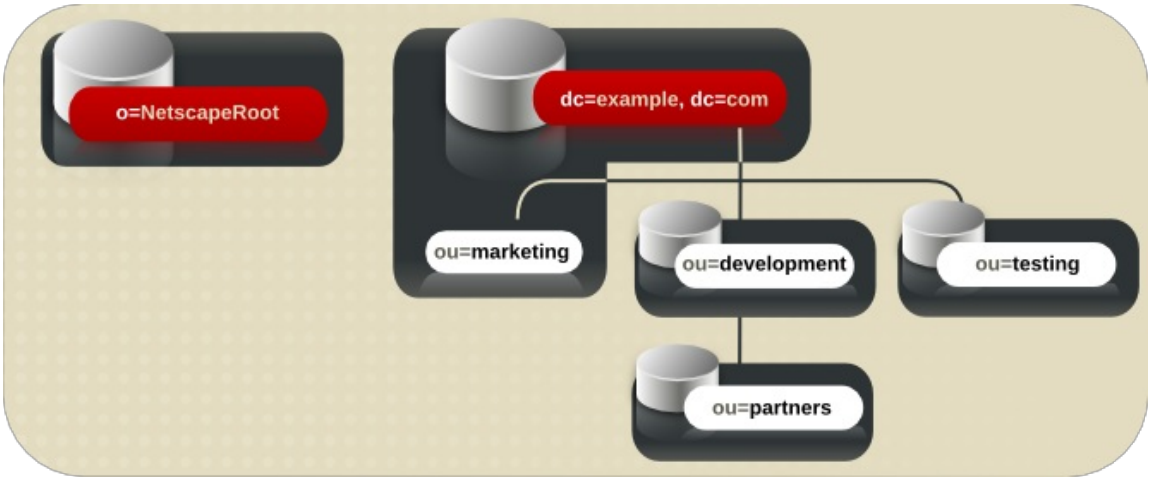


Figure 6.4. Directory Tree Spread across Multiple Databases

The resulting suffixes would contain the following entries:

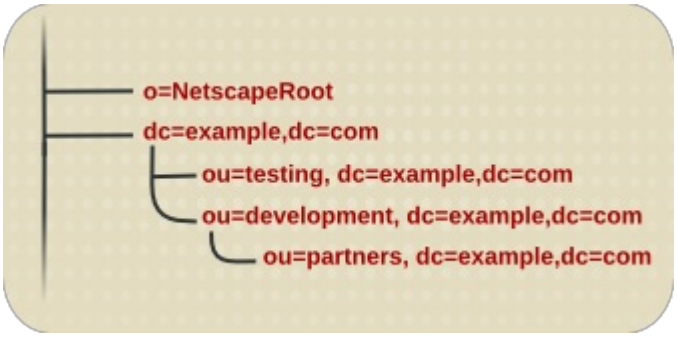


Figure 6.5. Suffixes for a Distributed Directory Tree

The **o=NetscapeRoot** and **dc=example,dc=com** suffixes are both root suffixes. The **ou=testing,dc=example,dc=com** suffix, the **ou=development,dc=example,dc=com** suffix, and the **ou=partners,ou=development,dc=example,dc=com** suffix are all subsuffixes of the **dc=example,dc=com** root suffix. The root suffix **dc=example,dc=com** contains the data in the **ou=marketing** branch of the original directory tree.

Using Multiple Root Suffixes

The directory service might contain more than one root suffix. For example, an ISP called "Example" might host several websites, one for example_a.com and one for example_b.com. The ISP would create two root suffixes, one

corresponding to the **o=example_a.com** naming context and one corresponding to the **o=example_b.com** naming context.

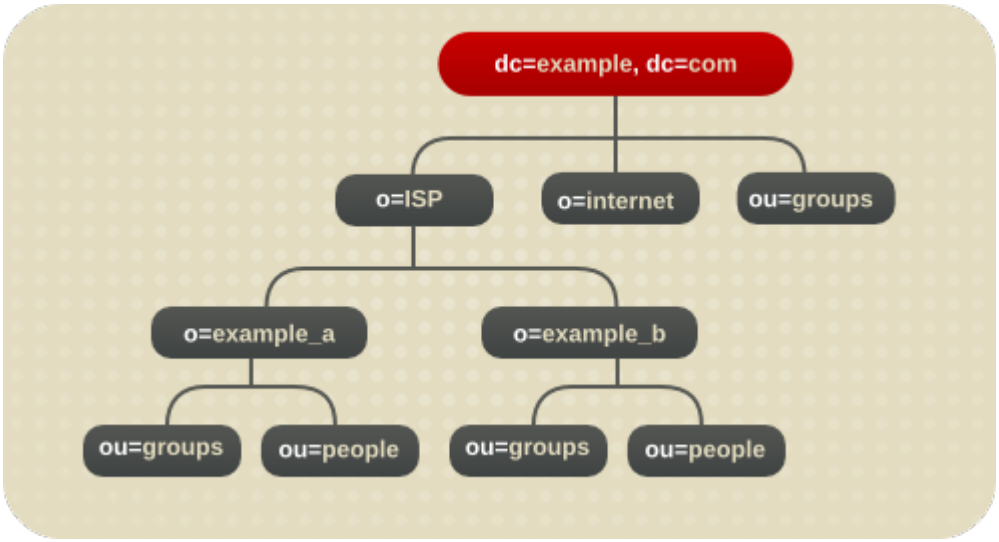


Figure 6.6. Directory Tree with Multiple Root Suffixes

The **dc=example,dc=com** entry represents a root suffix. The entry for each hosted ISP is also a root suffix (**o=example_a** and **o=example_b**). The **ou=people** and the **ou=groups** branches are subsuffixes under each root suffix.

6.3. ABOUT KNOWLEDGE REFERENCES

After distributing the data over several databases, define the relationship between the distributed data using *knowledge references*, pointers to directory information held in different databases. The Directory Server provides the following types of knowledge references to help link the distributed data into a single directory tree:

- Referrals – The server returns a piece of information to the client application indicating that the client application needs to contact another server to fulfill the request.
- Chaining – The server contacts other servers on behalf of the client application and returns the combined results to the client application when the operation is finished.

The following sections describe and compare these two types of knowledge references in more detail.

6.3.1. Using Referrals

A *referral* is a piece of information returned by a server that informs a client application which server to contact to proceed with an operation request. This redirection mechanism occurs when a client application requests a directory entry that does not exist on the local server.

Directory Server supports two types of referrals:

- Default referrals – The directory returns a default referral when a client application presents a DN for which the server does not have a matching suffix. Default referrals are stored in the configuration file of the server. One default referral can be set for the Directory Server and a separate default referral for each database.

  The default referral for each database is done through the suffix configuration information. When the suffix of the database is disabled, configure the directory service to return a default referral to client requests made to that suffix.

  For more information about suffixes, see Section 6.2.2, “About Suffixes”. For information on configuring suffixes, see the *Red Hat Directory Server Administrator's Guide* .

- Smart referrals – Smart referrals are stored on entries within the directory service itself. Smart referrals point to Directory Servers that have knowledge of the subtree whose DN matches the DN of the entry containing the smart referral.

All referrals are returned in the format of an LDAP uniform resource locator, or LDAP URL. The following sections describe the structure of an LDAP referral, and then describe the two referral types supported by Directory Server.

6.3.1.1. The Structure of an LDAP Referral

An LDAP referral contains information in the format of an LDAP URL. An LDAP URL contains the following information:

- The host name of the server to contact.
- The port number on the server that is configured to listen for LDAP requests.
- The base DN (for search operations) or target DN (for add, delete, and modify operations).

For example, a client application searches **dc=example,dc=com** for entries with a surname value of **Jensen**. A referral returns the following LDAP URL to the client application:

```
ldap://europe.example.com:389/ou=people, l=europe,dc=example,dc=com
```

This referral instructs the client application to contact the host **europe.example.com** on port 389 and submit a search using the root suffix **ou=people, l=europe,dc=example,dc=com**.

The LDAP client application determines how a referral is handled. Some client applications automatically retry the operation on the server to which they have been referred. Other client applications return the referral information to the user. Most LDAP client applications provided by Red Hat Directory Server (such as the command-line utilities) automatically follow the referral. The same bind credentials supplied on the initial directory request are used to access the server.

Most client applications follow a limited number of referrals, or *hops*. The limit on the number of referrals that are followed reduces the time a client application spends trying to complete a directory lookup request and helps eliminate hung processes caused by circular referral patterns.

6.3.1.2. About Default Referrals

Default referrals are returned to clients when the server or database that was contacted does not contain the requested data.

Directory Server determines whether a default referral should be returned by comparing the DN of the requested directory object against the directory suffixes supported by the local server. If the DN does not match the supported suffixes, the Directory Server returns a default referral.

For example, a directory client requests the following directory entry:
uid=bjensen,ou=people,dc=example,dc=com

However, the server only manages entries stored under the **dc=europe,dc=example,dc=com** suffix. The directory returns a referral to the client that indicates which server to contact for entries stored under the **dc=example,dc=com** suffix. The client then contacts the appropriate server and resubmits the original request.

Configure the default referral to point to a Directory Server that has more information about the distribution of the directory service. Default referrals for the server are set by the ***nsslapd-referral*** attribute. Default referrals for each database in the directory installation are set by the ***nsslapd-referral*** attribute in the database entry in the configuration. These attribute values are stored in the **dse.ldif** file.

For information on configuring default referrals, see the *Red Hat Directory Server Administrator's Guide* .

6.3.1.3. Smart Referrals

The Directory Server can also use *smart referrals*. Smart referrals associate a directory entry or directory tree to a specific LDAP URL. This means that requests can be forwarded to any of the following:

- The same namespace contained on a different server.
- Different namespaces on a local server.
- Different namespaces on the same server.

Unlike default referrals, smart referrals are stored within the directory service itself. For information on configuring and managing smart referrals, see the *Red Hat Directory Server Administrator's Guide* .

For example, the directory service for the American office of the Example Corp. contains the **ou=people,dc=example,dc=com** directory branch point.

Redirect all requests on this branch to the **ou=people** branch of the European office of Example Corp. by specifying a smart referral on the **ou=people** entry itself. The smart referral is **ldap://europe.example.com:389/ou=people,dc=example,dc=com**.

Any requests made to the people branch of the American directory service are redirected to the European directory. This is illustrated below:

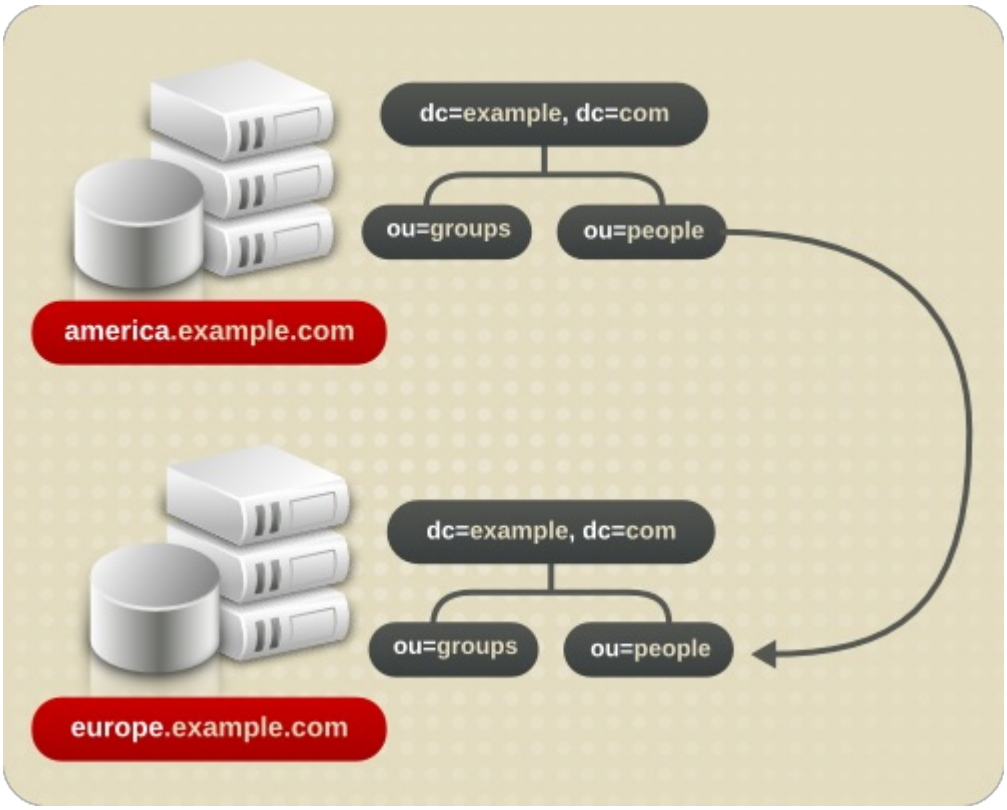


Figure 6.7. Using Smart Referrals to Redirect Requests

The same mechanism can be used to redirect queries to a different server that uses a different namespace. For example, an employee working in the Italian office of Example Corp. makes a request to the European directory service for the phone number of an Example Corp. employee in America. The directory service returns the referral **ldap://europe.example.com:389/ou=US employees,dc=example,dc=com**.

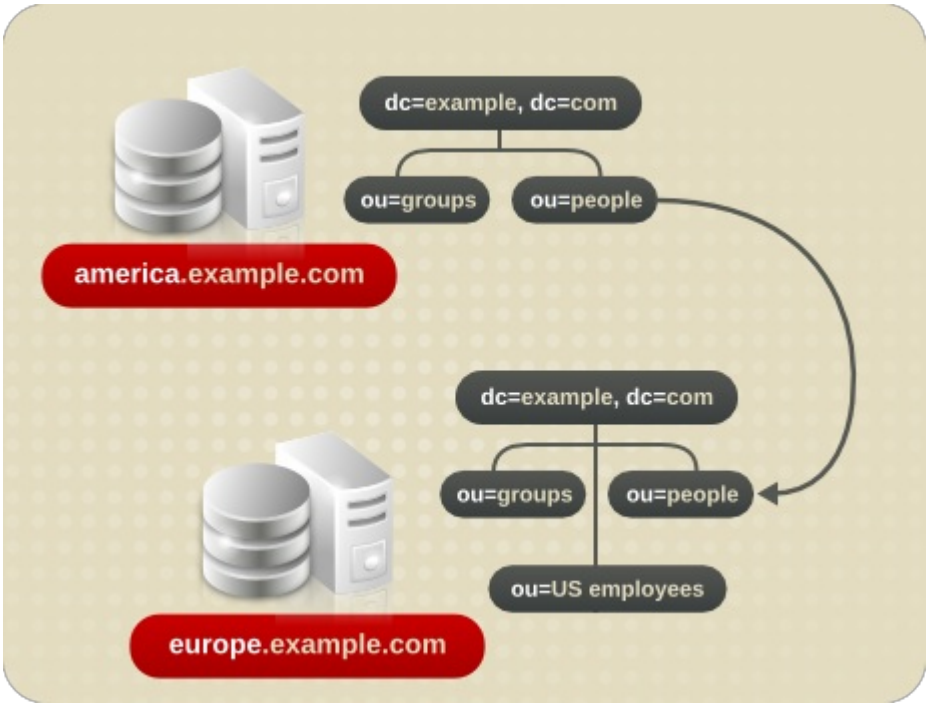


Figure 6.8. Redirecting a Query to a Different Server and Namespace

Finally, if multiple suffixes are served on the same server, queries can be redirected from one namespace to another namespace served on the same machine. For example, to redirect all queries on the local machine for **o=example,c=us** to **dc=example,dc=com**, then put the smart referral **ldap:///dc=example,dc=com** on the **o=example,c=us** entry.

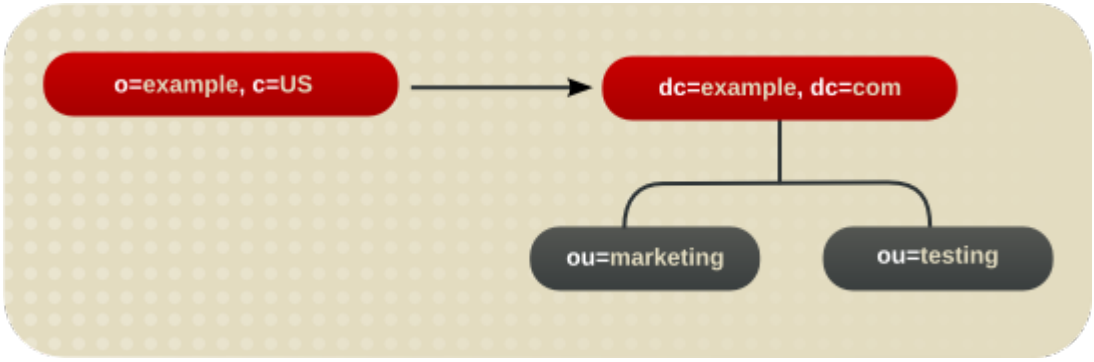


Figure 6.9. Redirecting a Query from One Namespace to Another Namespace on the Same Server

NOTE

The third slash in this LDAP URL indicates that the URL points to the same Directory Server.

Creating a referral from one namespace to another works only for clients whose searches are based at that distinguished name. Other kinds of operations, such as searches below **ou=people,o=example,c=US**, are not performed correctly.

For more information on LDAP URLS and on how to include smart URLs on Directory Server entries, see the *Red Hat Directory Server Administrator's Guide*.

6.3.1.4. Tips for Designing Smart Referrals

Even though smart referrals are easy to implement, consider the following points before using them:

- Keep the design simple.

  Deploying the directory service using a complex web of referrals makes administration difficult. Overusing smart referrals can also lead to circular referral patterns. For example, a referral points to an LDAP URL, which in turn points to another LDAP URL, and so on until a referral somewhere in the chain points back to the original server. This is illustrated below:

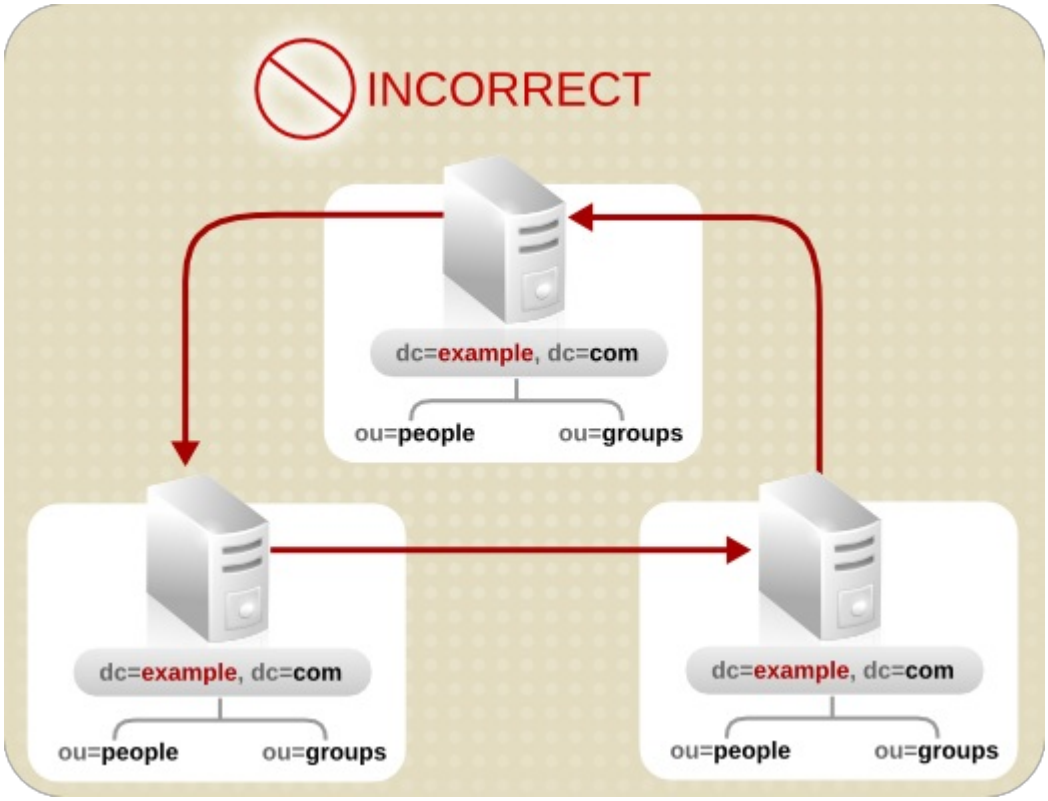


Figure 6.10. A Circular Referral Pattern

- Redirect at major branchpoints.

Limit referral usage to handle redirection at the suffix level of the directory tree. Smart referrals redirect lookup requests for leaf (non-branch) entries to different servers and DNs. As a result, it is tempting to use smart referrals as an aliasing mechanism, leading to a complex and difficult method to secure directory structure. Limiting referrals to the suffix or major branch points of the directory tree limits the number of referrals that have to be managed, subsequently reducing the directory's administrative overhead.

- Consider the security implications.

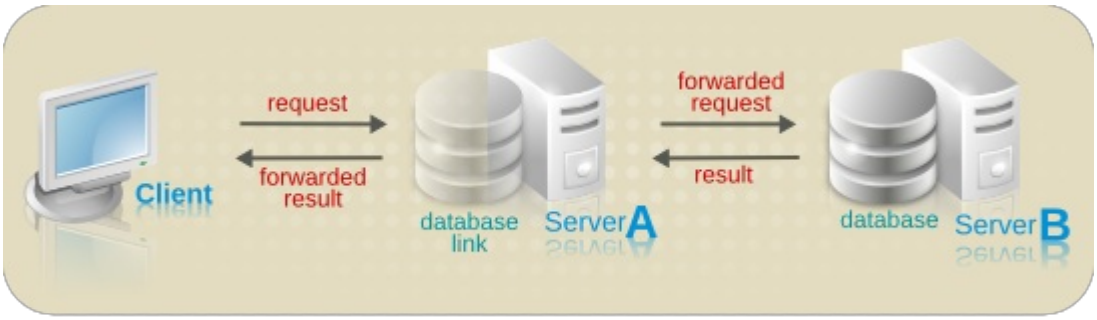
  Access control does not cross referral boundaries. Even if the server where the request originated allows access to an entry, when a smart referral sends a client request to another server, the client application may not be allowed access.

  In addition, the client's credentials need to be available on the server to which the client is referred for client authentication to occur.

6.3.2. Using Chaining

Chaining is a method for relaying requests to another server. This method is implemented through database links. A database link, as described in Section 6.2, "Distributing the Directory Data" , contains no data. Instead, it redirects client application requests to remote servers that contain the data.

During the chaining process, a server receives a request from a client application for data that the server does not contain. Using the database link, the server then contacts other servers on behalf of the client application and returns the results to the client application.



Each database link is associated with a remote server holding data. Configure alternate remote servers containing replicas of the data for the database link to use in the event of a failure. For more information on configuring database links, see the *Red Hat Directory Server Administrator's Guide* .

Database links provide the following features:

- Invisible access to remote data.

  Because the database link resolves client requests, data distribution is completely hidden from the client.

- Dynamic management.

  A part of the directory service can be added or removed from the system while the entire system remains available to client applications. The database link can temporarily return referrals to the application until entries have been redistributed across the directory service.

  This can also be implemented through the suffix itself, which can return a referral rather than forwarding a client application to the database.

- Access control.

  The database link impersonates the client application, providing the appropriate authorization identity to the remote server. User impersonation can be disabled on the remote servers when access control evaluation is not required. For more information on configuring database links, see the *Red Hat Directory Server Administrator's Guide*.

6.3.3. Deciding Between Referrals and Chaining

Both methods of linking the directory partitions have advantages and disadvantages. The method, or combination of methods, to use depends upon the specific needs of the directory service.

The major difference between the two knowledge references is the location of the intelligence that knows how to locate the distributed information. In a chained system, the intelligence is implemented in the servers. In a system that uses referrals, the intelligence is implemented in the client application.

While chaining reduces client complexity, it does so at the cost of increased server complexity. Chained servers must work with remote servers and send the results to directory clients.

With referrals, the client must handle locating the referral and collating search results. However, referrals offer more flexibility for the writers of client applications and allow developers to provide better feedback to users about the progress of a distributed directory operation.

The following sections describe some of the more specific differences between referrals and chaining in greater detail.

6.3.3.1. Usage Differences

Some client applications do not support referrals. Chaining allows client applications to communicate with a single server and still access the data stored on many servers. Sometimes referrals do not work when a company's network uses proxies. For example, a client application may have permissions to communicate with only one server inside a firewall. If that application is referred to a different server, it is not able to contact it successfully.

A client must also be able to authenticate correctly when using referrals, which means that the servers to which clients are being referred need to contain the client's credentials. With chaining, client authentication takes place only once. Clients do not need to authenticate again on the servers to which their requests are chained.

6.3.3.2. Evaluating Access Controls

Chaining evaluates access controls differently from referrals. With referrals, an entry for the client must exist on all of the target servers. With chaining, the client entry does not need to be on all of the target servers.

Performing Search Requests Using Referrals

The following diagram illustrates a client request to a server using referrals:

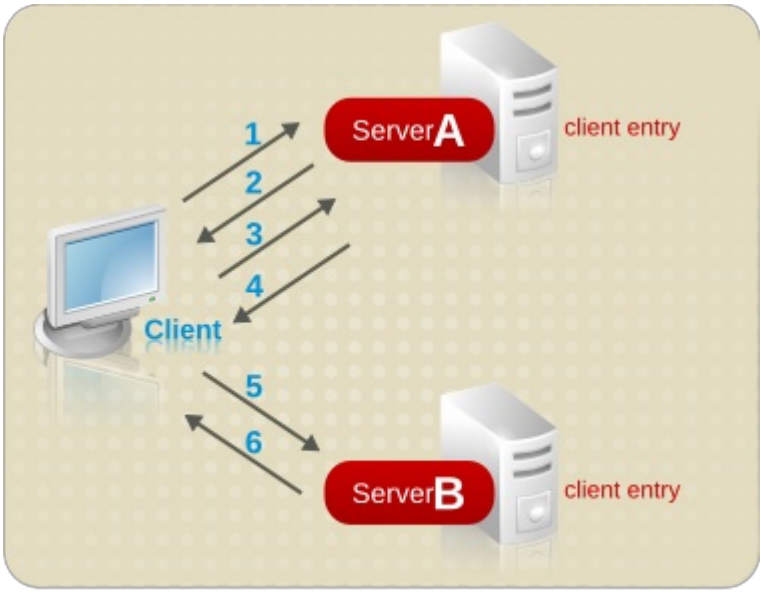


Figure 6.11. Sending a Client Request to a Server Using Referrals

In the illustration above, the client application performs the following steps:

1. The client application first binds with Server A.
2. Server A contains an entry for the client that provides a user name and password, so it returns a bind acceptance message. In order for the referral to work, the client entry must be present on server A.
3. The client application sends the operation request to Server A.
4. However, Server A does not contain the requested information. Instead, Server A returns a referral to the client application instructing it to contact Server B.
5. The client application then sends a bind request to Server B. To bind successfully, Server B must also contain an entry for the client application.
6. The bind is successful, and the client application can now resubmit its search operation to Server B.

This approach requires Server B to have a replicated copy of the client's entry from Server A.

Performing Search Requests Using Chaining

The problem of replicating client entries across servers is resolved using chaining. On a chained system, the search request is forwarded multiple times until there is a response.

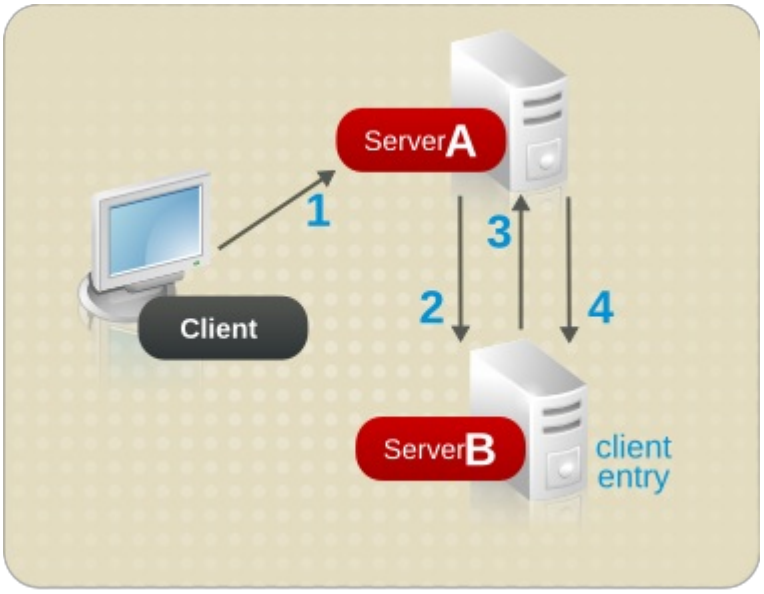


Figure 6.12. Sending a Client Request to a Server Using Chaining

In the illustration above, the following steps are performed:

1. The client application binds with Server A, and Server A tries to confirm that the user name and password are correct.
2. Server A does not contain an entry corresponding to the client application. Instead, it contains a database link to Server B, which contains the actual entry of the client. Server A sends a bind request to Server B.
3. Server B sends an acceptance response to Server A.
4. Server A then processes the client application's request using the database link. The database link contacts a remote data store located on Server B to process the search operation.

In a chained system, the entry corresponding to the client application does not need to be located on the same server as the data the client requests.

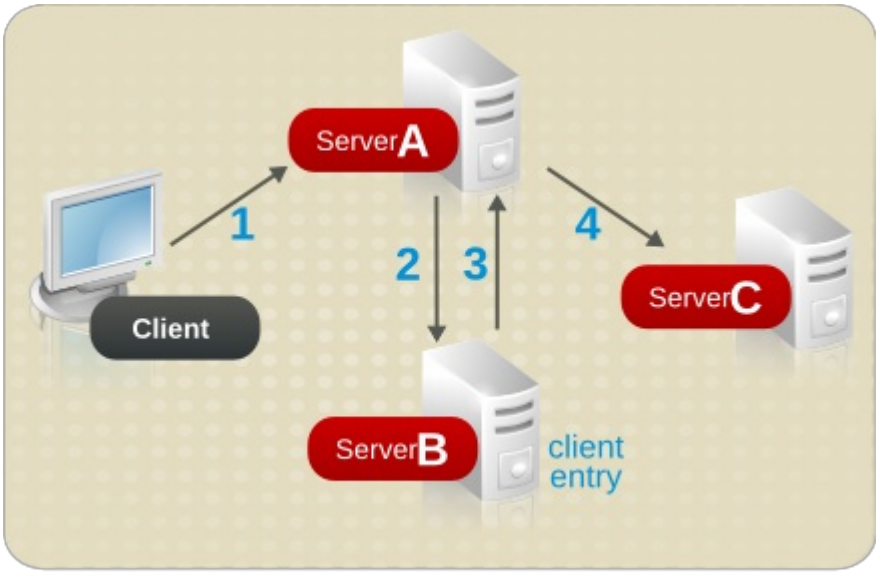


Figure 6.13. Authenticating a Client and Retrieving Data Using Different Servers

In this illustration, the following steps are performed:

1. The client application binds with Server A, and Server A tries to confirm that the user name and password are correct.
2. Server A does not contain an entry corresponding to the client application. Instead, it contains a database link to Server B, which contains the actual entry of the client. Server A sends a bind request to Server B.
3. Server B sends an acceptance response to Server A.
4. Server A then processes the client application's request using another database link. The database link contacts a remote data store located on Server C to process the search operation.

Unsupported Access Controls

Database links do not support the following access controls:

- Controls that must access the content of the user entry are not supported when the user entry is located on a different server. This includes access controls based on groups, filters, and roles.
- Controls based on client IP addresses or DNS domains may be denied. This is because the database link impersonates the client when it contacts remote servers. If the remote database contains IP-based access controls, it evaluates them using the database link's domain rather than the original client domain.

6.4. USING INDEXES TO IMPROVE DATABASE PERFORMANCE

Searches performed by client applications can be time and resource intensive, depending on the size of the databases. To help alleviate this problem, use indexes to improve search performance.

Indexes are files stored in the directory databases. Separate index files are maintained for each database in the directory service. Each file is named according to the attribute it indexes. The index file for a particular attribute can contain multiple types of indexes, so several types of index can be maintained for each attribute. For example, a file called **cn.db4** contains all of the indexes for the common name attribute.

Different types of indexes are used depending on the types of applications that use the directory service. Different applications may frequently search for a particular attribute, or may search the directory in a different language, or may require data in a particular format.

6.4.1. Overview of Directory Index Types

Directory Server supports the following types of index:

- Presence index – Lists entries that possess a particular attribute, such as ***uid***.
- Equality index – Lists entries that contain a specific attribute value, such as **cn=Babs Jensen**.
- Approximate index – Allows approximate (or "sounds-like") searches. For example, an entry might contain the attribute value of **cn=Babs L. Jensen**. An approximate search would return this value for searches against **cn~=Babs Jensen**, **cn~=Babs**, and **cn~=Jensen**.

  NOTE

  Approximate indexes require that names be written in English using ASCII characters.

- Substring index – Allows searches against substrings within entries. For example, a search for **cn=*derson** would match common names containing this string (such as Bill Anderson, Norma Henderson, and Steve Sanderson).
- International index – Improves the performance of searches for information in international directories. Configure the index to apply a matching rule by associating a locale (internationalization OID) with the attribute being indexed.
- Browsing index or virtual list view (VLV) index – Improves the display performance of entries in the Directory Server Console. A *browsing index* can be created on any branch in the directory tree to improve the display performance.

6.4.2. Evaluating the Costs of Indexing

Indexes improve search performance in the directory databases, but there is a cost involved:

- Indexes increase the time it takes to modify entries.

  The more indexes being maintained, the longer it takes the directory service to update the database.

- Index files use disk space.

  The more attributes being indexed, the more files are created. If there are approximate and substring indexes for attributes that contain long strings, these files can grow rapidly.

- Index files use memory.

  To run more efficiently, the directory service places as many index files in memory as possible. Index files use memory out of the pool available depending upon the database cache size. A large number of index files requires a larger database cache.

- Index files take time to create.

  Although index files save time during searches, maintaining unnecessary indexes can waste time. Be certain to maintain only the files needed by the client applications using the directory service.

CHAPTER 7. DESIGNING THE REPLICATION PROCESS

Replicating the directory contents increases the availability and performance of the directory service. Chapter 4, *Designing the Directory Tree* and Chapter 6, *Designing the Directory Topology* cover the design of the directory tree and the directory topology. This chapter addresses the physical and geographical location of the data and, specifically, how to use replication to ensure the data is available when and where it is needed.

This chapter discusses uses for replication and offers advice on designing a replication strategy for the directory environment.

7.1. INTRODUCTION TO REPLICATION

Replication is the mechanism that automatically copies directory data from one Red Hat Directory Server to another. Using replication, any directory tree or subtree (stored in its own database) can be copied between servers. The Directory Server that holds the master copy of the information automatically copies any updates to all replicas.

Replication provides a high-availability directory service and can distribute the data geographically. In practical terms, replication provides the following benefits:

- Fault tolerance and failover — By replicating directory trees to multiple servers, the directory service is available even if hardware, software, or network problems prevent the directory client applications from accessing a particular Directory Server. Clients are referred to another Directory Server for read and write operations.

  NOTE

  Write failover is only possible with multi-master replication.

- Load balancing — Replicating the directory tree across servers reduces the access load on any given machine, thereby improving server response time.
- Higher performance and reduced response times — Replicating directory entries to a location close to users significantly improves directory response times.
- Local data management — Replication allows information to be owned and managed locally while sharing it with other Directory Servers across the enterprise.

7.1.1. Replication Concepts

Always start planning replication by making the following fundamental decisions:

- What information to replicate.
- Which servers hold the master copy, or *read-write replica*, of that information.
- Which servers hold the read-only copy, or *read-only replica*, of that information.
- What should happen when a read-only replica receives an update request; that is, to which server it should refer the request.

These decisions cannot be made effectively without an understanding of how the Directory Server handles these concepts. For example, decide what information to replicate, be aware of the smallest replication unit that the Directory Server can handle. The replication concepts used by the Directory Server provide a framework for thinking about the global decisions that need to be made.

7.1.1.1. Unit of Replication

The smallest unit of replication is a database. An entire database can be replicated but not a subtree within a database. Therefore, when defining the directory tree, always consider replication. For more information on how to set up the directory tree, see Chapter 4, *Designing the Directory Tree*.

The replication mechanism also requires that one database correspond to one suffix. A suffix (or namespace) that is distributed over two or more databases cannot be replicated.

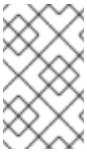
7.1.1.2. Read-Write and Read-Only Replicas

A database that participates in replication is defined as a *replica*. Directory Server supports two types of replicas: read-write and read-only. The read-write replicas contain master copies of directory information and can be updated. Read-only replicas refer all update operations to read-write replicas.

7.1.1.3. Suppliers and Consumers

A server that stores a replica that is copied to a different server is called a *supplier*. A server that stores a replica that is copied from a different server is called a *consumer*. Generally speaking, the replica on the supplier server is a read-write replica; the replica on the consumer server is a read-only replica. However, the following exceptions apply:

- In the case of *cascading replication*, the *hub supplier* holds a read-only replica that it supplies to consumers. For more information, see Section 7.2.3, "Cascading Replication".
- In the case of *multi-master replication*, the suppliers function as both suppliers and consumers for the same read-write replica. For more information, see Section 7.2.2, "Multi-Master Replication".



NOTE

In the current version of Red Hat Directory Server, replication is always initiated by the supplier server, never by the consumer. This is unlike earlier versions of Directory Server, which allowed consumer-initiated replication (where consumer servers could retrieve data from a supplier server).

Suppliers

For any particular replica, the supplier server must:

- Respond to read requests and update requests from directory clients.
- Maintain state information and a changelog for the replica.
- Initiate replication to consumer servers.

The supplier server is always responsible for recording the changes made to the read-write replicas that it manages, so the supplier server makes sure that any changes are replicated to consumer servers.

Consumers

A consumer server must:

- Respond to read requests.
- Refer update requests to a supplier server for the replica.

Whenever a consumer server receives a request to add, delete, or change an entry, the request is referred to a supplier for the replica. The supplier server performs the request, then replicates the change.

Hub Suppliers

In the special case of cascading replication, the hub supplier must:

- Respond to read requests.
- Refer update requests to a supplier server for the replica.
- Initiate replication to consumer servers.

For more information on cascading replication, see Section 7.2.3, "Cascading Replication".

7.1.1.4. Replication and Changelogs

Every supplier server maintains a *changelog*. A changelog is a record of the modifications that have occurred on a replica. The supplier server then replays these modifications on the replicas stored on consumer servers, or on other suppliers in the case of multi-master replication.

When an entry is modified, a change record describing the LDAP operation that was performed is recorded in the changelog.

The changelog size is maintained with two attributes, ***nsslapd-changelogmaxage*** or ***nsslapd-changelogmaxentries***. These attributes trim the old changelogs to keep the changelog size reasonable.

7.1.1.5. Replication Agreement

Directory Servers use replication agreements to define replication. A replication agreement describes replication between a single supplier and a single consumer. The agreement is configured on the supplier server. It identifies:

- The database to replicate.

- The consumer server to which the data is pushed.
- The times that replication can occur.
- The DN that the supplier server must use to bind (called the *supplier bind DN*).
- How the connection is secured (TLS/SSL, Start TLS, client authentication, SASL, or simple authentication).
- Any attributes that will not be replicated (see Section 7.3.2, "Replicate Selected Attributes with Fractional Replication").

7.1.2. Data Consistency

Consistency refers to how closely the contents of replicated databases match each other at a given point in time. Part of the configuration for replication between servers is to schedule updates. The supplier server always determines when consumer servers need to be updated and initiates replication.

Directory Server offers the option of keeping replicas always synchronized or of scheduling updates for a particular time of day or day in the week.

The advantage of keeping replicas constantly synchronized is that it provides better data consistency. The cost is the network traffic resulting from the frequent update operations. This solution is the best option when:

- There is a reliable, high-speed connection between servers.
- The client requests serviced by the directory service are mainly search, read, and compare operations, with relatively few update operations.

If it is all right to a lower level of data consistency, choose the frequency of updates that best suits the use patterns of the network or lowers the affect on network traffic. There are several situations where having scheduled updates instead of constant updates is the best solution:

- There are unreliable or intermittently available network connections.
- The client requests serviced by the directory service are mainly update operations.
- Communication costs have to be lowered.

In the case of multi-master replication, the replicas on each supplier are said to be *loosely consistent*, because at any given time, there can be differences in the data stored on each supplier. This is true, even if the replicas are constantly synchronized, for two reasons:

- There is a latency in the propagation of update operations between suppliers.
- The supplier that serviced the update operation does not wait for the second supplier to validate it before returning an "operation successful" message to the client.

7.2. COMMON REPLICATION SCENARIOS

Decide how the updates flow from server to server and how the servers interact when propagating updates. There are the four basic scenarios and a few strategies for deciding the method appropriate for the environment. These basic scenarios can be combined to build the replication topology that best suits the network environment.

- Section 7.2.1, "Single-Master Replication"
- Section 7.2.2, "Multi-Master Replication"
- Section 7.2.3, "Cascading Replication"
- Section 7.2.4, "Mixed Environments"

7.2.1. Single-Master Replication

In the most basic replication configuration, a supplier server copies a replica directly to one or more consumer servers. In this configuration, all directory modifications occur on the read-write replica on the supplier server, and the consumer servers contain read-only replicas of the data.

The supplier server must perform all modifications to the read-write replicas stored on the consumer servers. This is illustrated below.

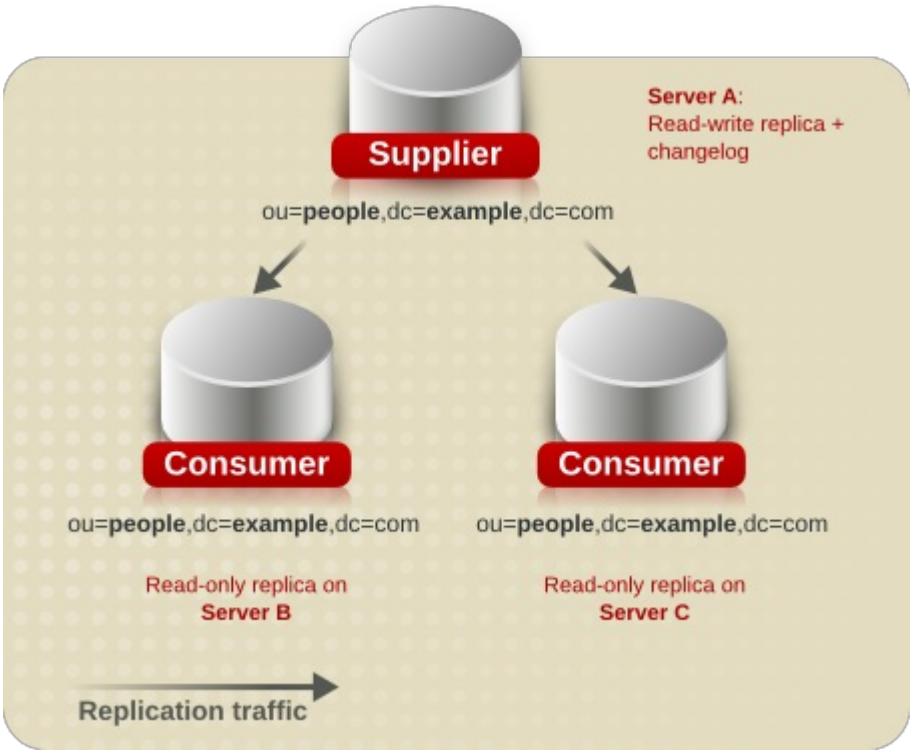


Figure 7.1. Single-Master Replication

The supplier server can replicate a read-write replica to several consumer servers. The total number of consumer servers that a single supplier server can manage depends on the speed of the networks and the total number of entries that are modified on a daily basis. However, a supplier server is capable of maintaining several consumer servers.

7.2.2. Multi-Master Replication

In a *multi-master replication* environment, master copies of the same information can exist on multiple servers. This means that data can be updated simultaneously in different locations. The changes that occur on each server are replicated to the other servers. This means that each server functions as both a supplier and a consumer.

NOTE

Red Hat Directory Server supports a maximum of 20 master servers in any replication environment, as well as an unlimited number of hub suppliers. The number of consumer servers that hold the read-only replicas is unlimited.

When the same data is modified on multiple servers, there is a conflict resolution procedure to determine which change is kept. The Directory Server considers the valid change to be the most recent one.

Multiple servers can have master copies of the same data, but, within the scope of a single replication agreement, there is only one supplier server and one consumer. Consequently, to create a multi-master environment between two supplier servers that share responsibility for the same data, create more than one replication agreement.

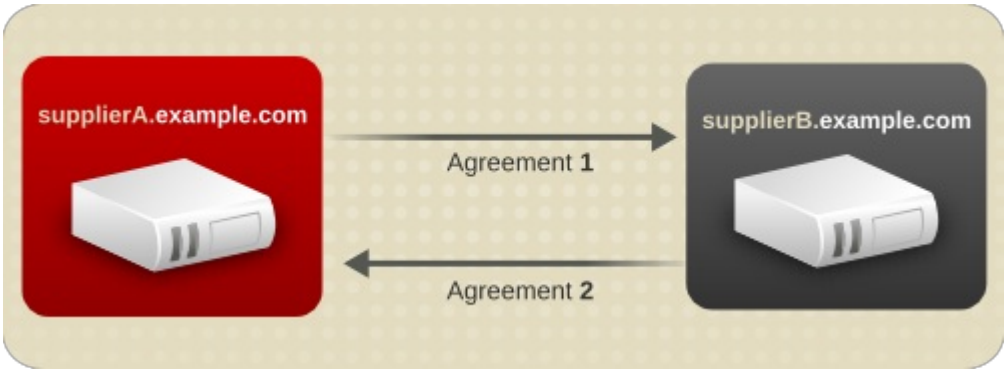


Figure 7.2. Simplified Multi-Master Replication Configuration

In Figure 7.2, "Simplified Multi-Master Replication Configuration", supplier A and supplier B each hold a read-write replica of the same data.

Figure 7.3, "Replication Traffic in a Simple Multi-Master Environment" illustrates the replication traffic with two suppliers (read-write replicas in the illustration), and two consumers (read-only replicas in the illustration). The consumers can be updated by both suppliers. The supplier servers ensure that the changes do not collide.

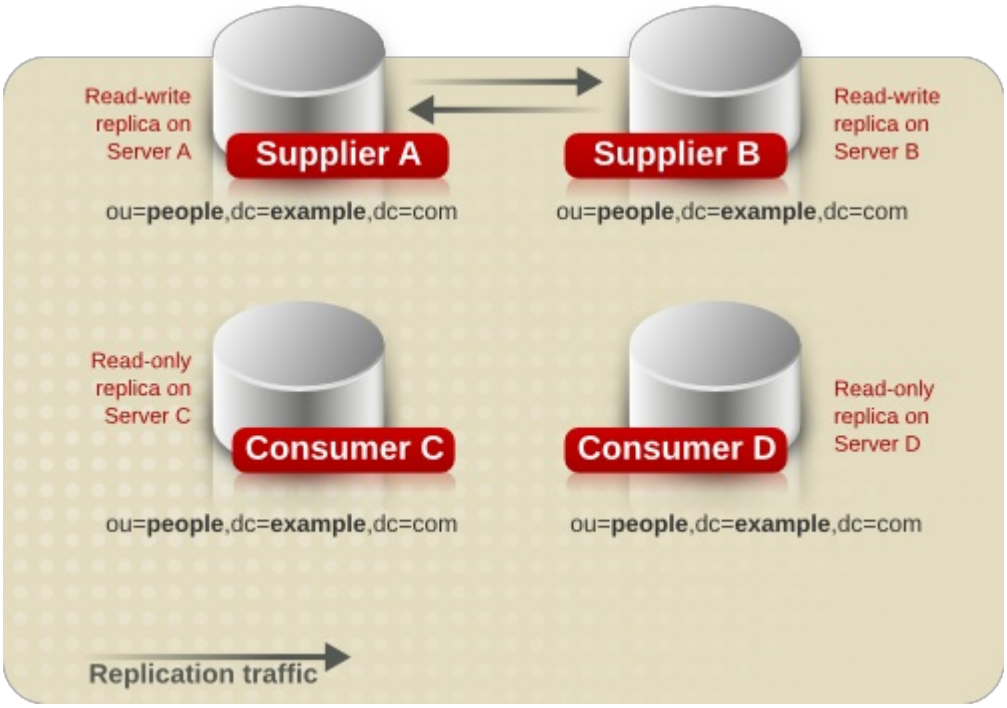


Figure 7.3. Replication Traffic in a Simple Multi-Master Environment

Replication in Directory Server can support as many as 20 masters, which all share responsibility for the same data. Using that many suppliers requires creating a range of replication agreements. (Also remember that in multi-master replication, each of the suppliers can configured in different topologies — meaning there can be 20 different directory trees and even schema differences. There are many variables that have a direct impact on the topology selection.)

In multi-master replication, the suppliers can send updates to all other suppliers or to some subset of other suppliers. Sending updates to all other suppliers means that changes are propogated faster and the overall scenario has much better failure-tolerance. However, it also increases the complexity of configuring suppliers and introduces high network demand and high server demand. Sending updates to a subset of suppliers is much simpler to configure and reduces the network and server loads, but there is a risk that data could be lost if there were multiple server failures.

Figure 7.4, "Multi-Master Replication Configuration A" illustrates a fully connected mesh topology where four supplier servers feed data to the other three supplier servers (which also function as consumers). A total of twelve replication agreements exist between the four supplier servers.

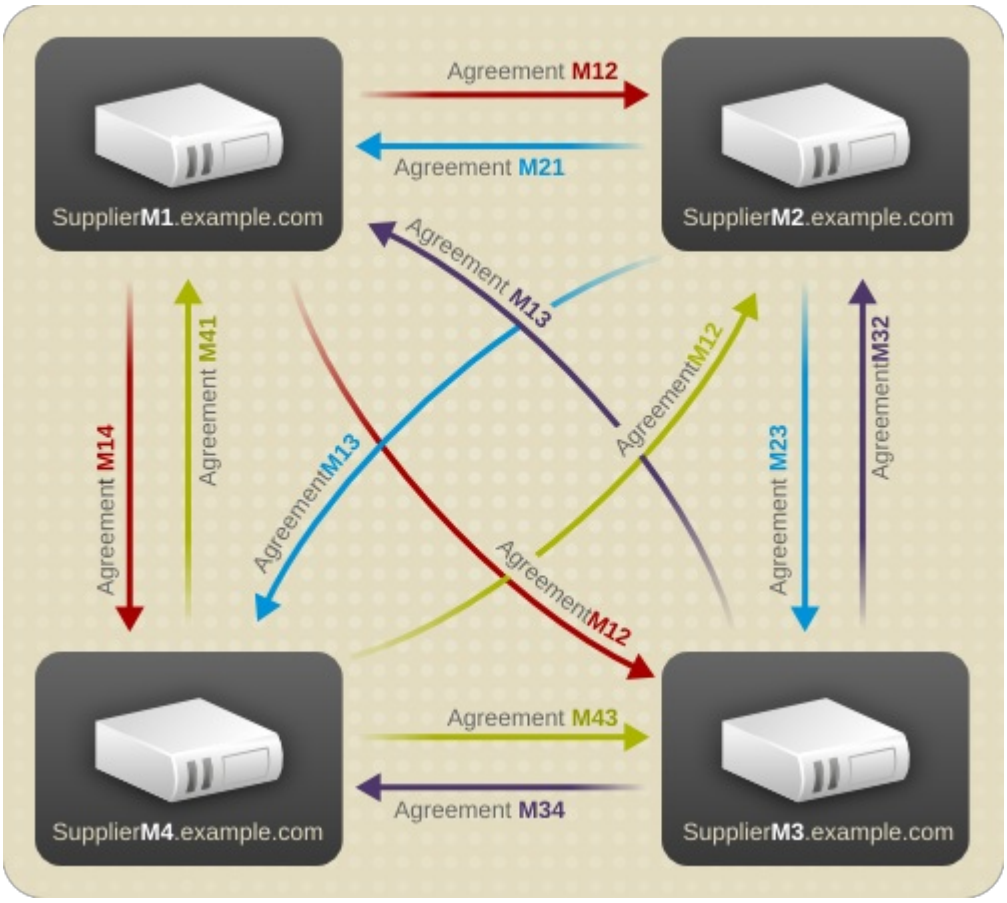


Figure 7.4. Multi-Master Replication Configuration A

Figure 7.5, "Multi-Master Replication Configuration B" illustrates a topology where each supplier server feeds data to two other supplier servers (which also function as consumers). Only eight replication agreements exist between the four supplier servers, compared to the twelve agreements shown for the topology in Figure 7.4, "Multi-Master Replication Configuration A". This topology is beneficial where the possibility of two or more servers failing at the same time is negligible.

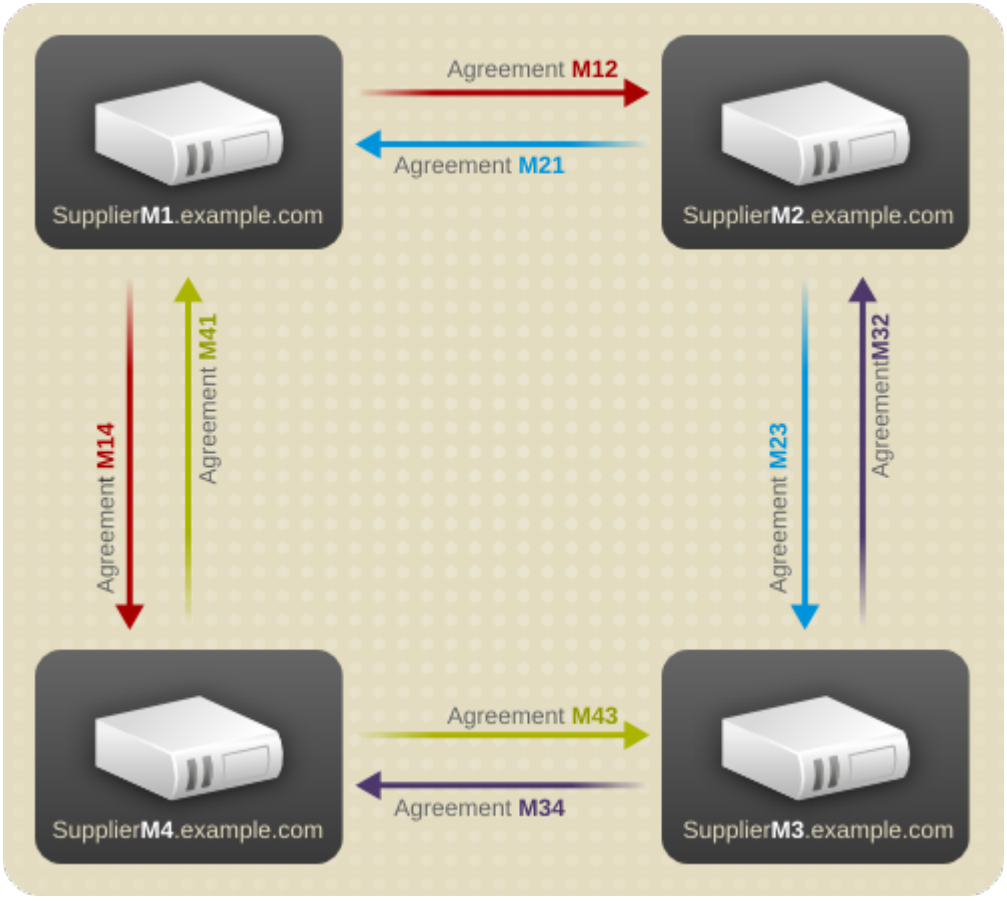


Figure 7.5. Multi-Master Replication Configuration B

Those two examples are simplified multi-master scenarios. Since Red Hat Directory Server can have as many as 20 masters and an unlimited number of hub suppliers in a single multi-master environment, the replication topology can become much more complex. For example, Figure 7.4, "Multi-Master Replication Configuration A" has 12 replication agreements (four suppliers with three agreements each). If there are 20 masters, then there are 380 replication agreements (20 servers with 19 agreements each).

WHen planning multi-master replication, consider:

- How many suppliers there will be
- What their geographic locations are
- The path the suppliers will use to update servers in other locations
- The topologies, directory trees, and schemas of the different suppliers
- The network quality
- The server load and performance
- The update interval required for directory data

7.2.3. Cascading Replication

In a cascading replication scenario, a *hub supplier* receives updates from a supplier server and replays those updates on consumer servers. The hub supplier is a hybrid; it holds a read-only replica, like a typical consumer server, and it also maintains a changelog like a typical supplier server.

Hub suppliers forward master data as they receive it from the original suppliers. Similarly, when a hub supplier receives an update request from a directory client, it refers the client to the supplier server.

Cascading replication is useful if some of the network connections between various locations in the organization are better than others. For example, Example Corp. keeps the master copy of its directory data in Minneapolis, and the consumer servers in New York and Chicago. The network connection between Minneapolis and New York is very good, but the connection between Minneapolis and Chicago is poor. Since the network between New York and Chicago is fair, Example administrators use cascading replication to move directory data from Minneapolis to New York to Chicago.

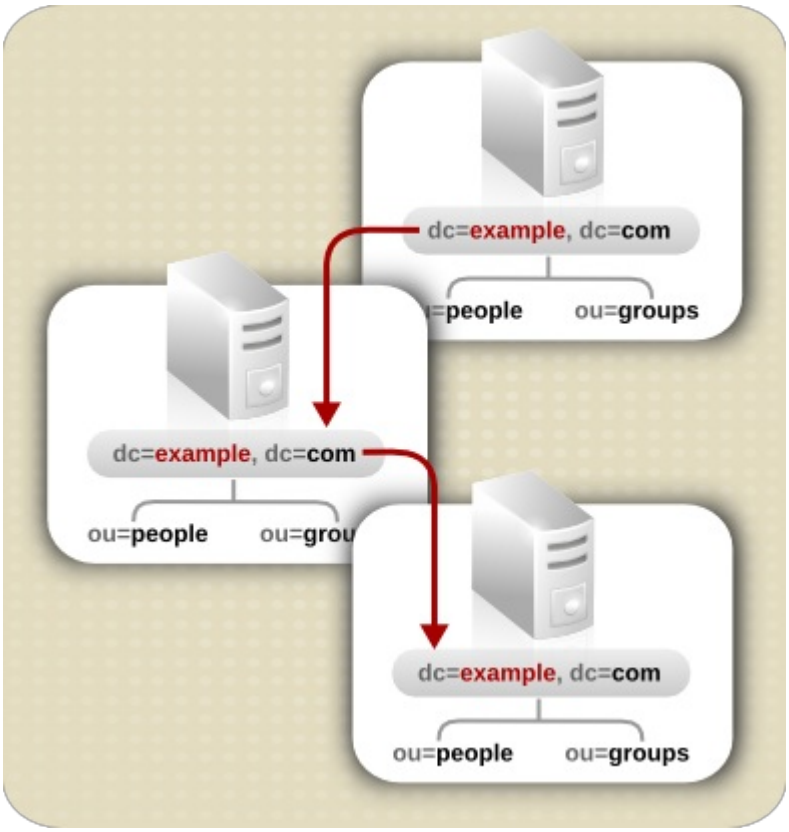


Figure 7.6. Cascading Replication Scenario

Figure 7.7, "Replication Traffic and Changelogs in Cascading Replication" illustrates the same scenario from a different perspective, which shows how the replicas are configured on each server (read-write or read-only), and which servers maintain a changelog.

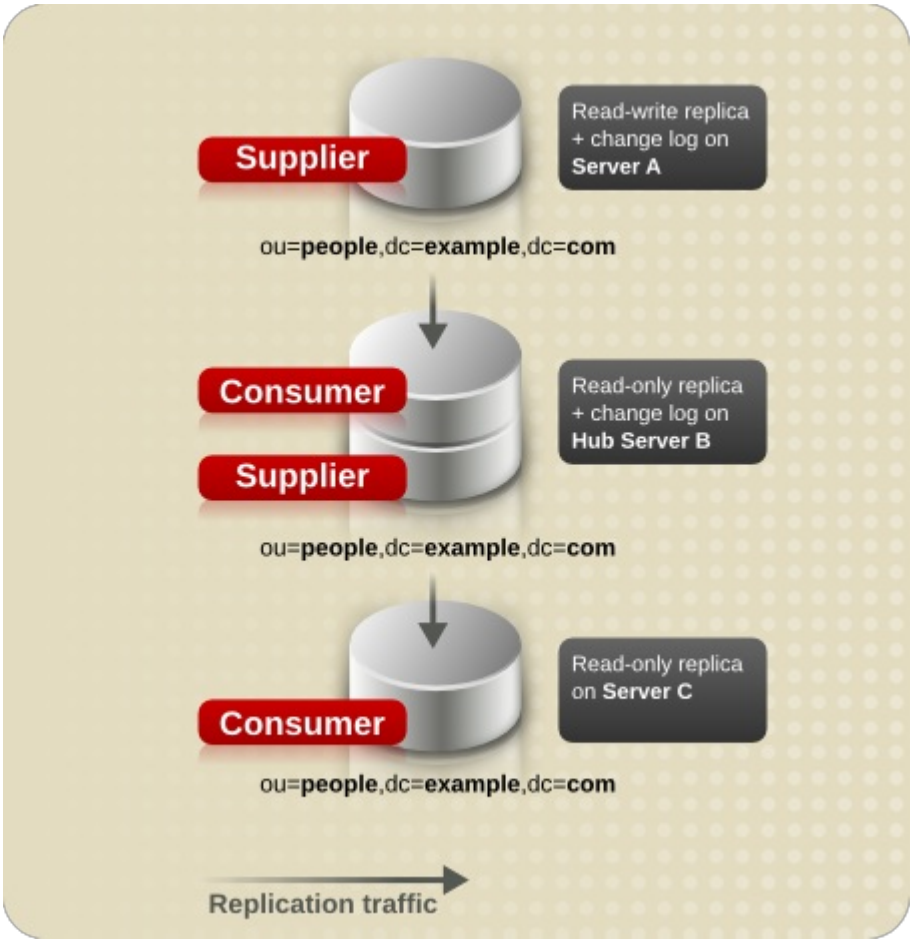


Figure 7.7. Replication Traffic and Changelogs in Cascading Replication

7.2.4. Mixed Environments

Any of the replication scenarios can be combined to meet suit the needs of the network and directory environment. One common combination is to use a multi-master configuration with a cascading configuration.

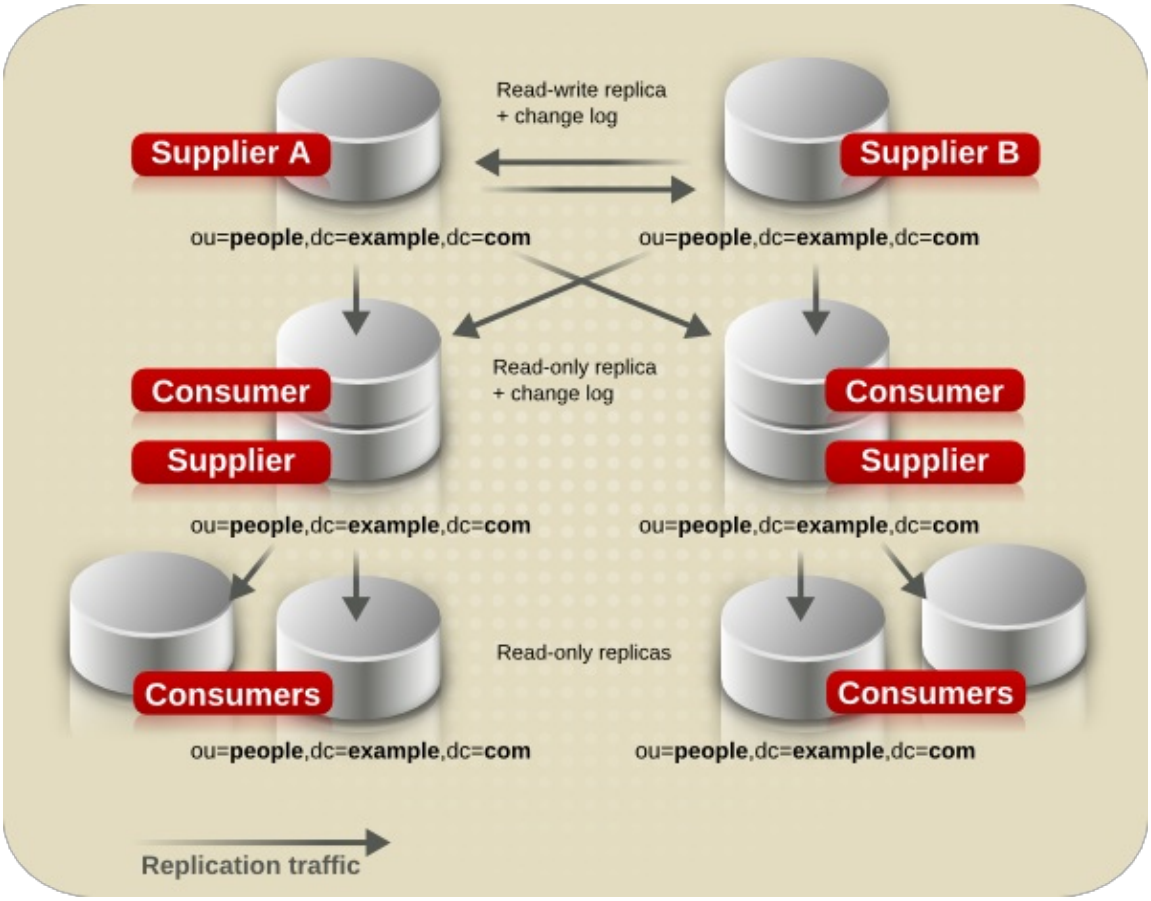


Figure 7.8. Combined Multi-Master and Cascading Replication

7.3. DEFINING A REPLICATION STRATEGY

The replication strategy is determined by the services that must be provided. To determine the replication strategy, start by performing a survey of the network, users, applications, and how they use the directory service.

- Assess the resources within the network, the traffic loads, and resource requirements for the directory service.

  See Section 7.3.1, "Conducting a Replication Survey", Section 7.3.3, "Replication Resource Requirements", and Section 7.3.4, "Managing Disk Space Required for Multi-Master Replication".

- If there are multiple consumers for different locations or sections of the company or if some servers are insecure, then use *fractional replication* to exclude sensitive or seldom-modified information to maintain data integrity without compromising sensitive information.

  See Section 7.3.2, "Replicate Selected Attributes with Fractional Replication" for more information.

- If the network is stretched across a wide geographical area, there are multiple Directory Servers at multiple sites, with local data masters connected by multi-master replication.

  See Section 7.3.5, "Replication Across a Wide-Area Network" for more information.

- If high availability is the primary concern, create a data center with multiple Directory Servers on a single site. Single-master replication provides read-failover, while multi-master replication provides write-failover.

  See Section 7.3.6, "Using Replication for High Availability" for more information.

- If local availability is the primary concern, use replication to distribute data geographically to Directory Servers in local offices around the world. A master copy of all information can be maintained in a single location, such as the company headquarters, or each local site can manage the parts of the DIT that are relevant for them.

See Section 7.3.7, “Using Replication for Local Availability” for more information.

- In all cases, balance the load of requests serviced by the Directory Servers and avoid network congestion.

  See Section 7.3.8, “Using Replication for Load Balancing” for more information.

After planning the replication strategy, it is possible to deploy the directory service. It is best to deploy the directory service in stages, because this allows administrators to adjust the directory service according to the loads that the enterprise places on the directory service. Unless the load analysis is based on an already operating directory, be prepared to alter the directory services as the real-life demands on the directory become clear.

7.3.1. Conducting a Replication Survey

Gather information about the network quality and usage in the site survey to help define the replication strategy:

- The quality of the LANs and WANs connecting different buildings or remote sites and the amount of available bandwidth.
- The physical location of users, how many users are at each site, and their usage patterns; that is how they intend to use the directory service.
- The number of applications that access the directory service and the relative percentage of read, search, and compare operations to write operations.
- If the messaging server uses the directory, find out how many operations it performs for each email message it handles. Other products that rely on the directory service are typically products such as authentication applications or meta-directory applications. For each one, determine the type and frequency of operations that are performed in the directory service.
- The number and size of the entries stored in the directory service.

A site that manages human resource databases or financial information is likely to put a heavier load on the directory service than a site containing engineering staff that uses the directory solely for telephone book purposes.

7.3.2. Replicate Selected Attributes with Fractional Replication

Fractional replication allows the administrator to choose a set of attributes that are not transmitted from a supplier to the consumer (or another supplier). Administrators can therefore replicate a database without replicating all the information that it contains.

Fractional replication is enabled and configured per replication agreement. The exclusion of attributes is applied equally to all entries. As far as the consumer server is concerned, the excluded attributes always have no value. Therefore, a client performing a search against the consumer server never sees the excluded attributes. Similarly, should it perform a search that specifies those attributes in its filter, no entries match.

Fractional replication is particularly useful in the following situations:

- Where the consumer server is connected using a slow network, excluding infrequently changed attributes or larger attributes such as ***jpegPhoto*** results in less network traffic.
- Where the consumer server is placed on an untrusted network such as the public Internet, excluding sensitive attributes such as telephone numbers provides an extra level of protection that guarantees no access to those attributes even if the server's access control measures are defeated or the machine is compromised by an attacker.

Configuring fractional replication is described in the replication agreement and supplier configuration sections in chapter 8, "Managing Replication," in the *Administrator's Guide*.

7.3.3. Replication Resource Requirements

Using replication requires more resources. Consider the following resource requirements when defining the replication strategy:

- Disk usage – On supplier servers, the changelog is written after each update operation. Supplier servers that receive many update operations may experience higher disk usage.

  NOTE

  Each supplier server uses a single changelog. If a supplier contains multiple replicated databases, the changelog is used more frequently, and the disk usage is even higher.

- Server threads — Each replication agreement consumes one server thread. So, the number of threads available to client applications is reduced, possibly affecting the server performance for the client applications.

- File descriptors — The number of file descriptors available to the server is reduced by the changelog (one file descriptor) and each replication agreement (one file descriptor per agreement).

7.3.4. Managing Disk Space Required for Multi-Master Replication

Multi-master replicas maintain additional logs, including the changelog of directory edits, state information for update entries, and tombstone entries for deleted entries. This information is required for multi-master replication to be performed. Because these log files can get very large, periodically cleaning up these files is necessary to keep from wasting disk space.

There are four attributes which can configure the changelog maintenance for the multi-master replica. Two are under **cn=changelog5** and relate directly to trimming the changelog:

- ***nsslapd-changelogmaxage*** sets the maximum age that the entries in the changelog can be; once an entry is older than that limit, it is deleted. This keeps the changelog from growing indefinitely.

- ***nsslapd-changelogmaxentries*** sets the maximum number of entries that are allowed in the changelog. Like ***nsslapd-changelogmaxage***, this also trims the changelog, but be careful about the setting. This must be large enough to allow a complete set of directory information or multi-master replication may not function properly.

The other two attributes are under the replication agreement entry in **cn=replica, cn=***suffixDN***, cn=mapping tree, cn=config**. These two attributes relate to maintenance information kept in the changelog, the tombstone and state information, rather than the directory edits information.

- ***nsDS5ReplicaPurgeDelay*** sets the maximum age that tombstone (deleted) entries and state information can be in the changelog. Once a tombstone or state information entry is older than that age, it is deleted. This differs from the ***nsslapd-changelogmaxage*** attribute in that the ***nsDS5ReplicaPurgeDelay*** value applies only to tombstone and state information entries; ***nsslapd-changelogmaxage*** applies to every entry in the changelog, including directory modifications.

- ***nsDS5ReplicaTombstonePurgeInterval*** sets the frequency which the server runs a purge operation. At this interval, the Directory Server runs an internal operation to clean the tombstone and state entries out of the changelog. Make sure that the maximum age is longer than the longest replication update schedule or multi-master replication may not be able to update replicas properly.

The parameters for managing replication and the changelog are described in chapter 2, "Core Configuration Attributes," in the *Configuration, Command, and File Reference* .

7.3.5. Replication Across a Wide-Area Network

Wide-area networks typically have higher latency, a higher bandwidth-delay product, and lower speeds than local area networks . Directory Server supports efficient replication when a supplier and consumer are connected using a wide-area network.

In previous versions of Directory Server, the replication protocols that were used to transmit entries and updates between suppliers and consumers were highly latency-sensitive, because the supplier would send only one update operation and then wait for a response from the consumer. This led to reduced throughput with higher latencies.

The supplier sends many updates and entries to the consumer without waiting for a response. Thus, on a network with high latency, many replication operations can be in transit on the network, and replication throughput is similar to that which can be achieved on a local area network.

NOTE

If a supplier is connected to another supplier running a version of Red Hat Directory Server earlier than 7.1, it falls back to the old replication mechanism for compatibility. It is therefore necessary to run at least version 7.1 on both the supplier and consumer servers in order to achieve latency-insensitive replication.

There are both performance and security issues to consider for both the Directory Server and the efficiency of the network connection:

- Where replication is performed across a public network such as the Internet, the use of SSL is highly recommended. This guards against eavesdropping of the replication traffic.

- Use a T-1 or faster Internet connection for the network.

- When creating agreements for replication over a wide-area network, avoid constant synchronization between the servers. Replication traffic could consume a large portion of the bandwidth and slow down the overall network and Internet connections.

- When initializing consumers, do not to initialize the consumer immediately; instead, utilize file system replica initialization, which is much faster than online initialization or initializing from file. See the *Red Hat Directory Server Administrator's Guide* for information on using filesystem replica initialization.

7.3.6. Using Replication for High Availability

Use replication to prevent the loss of a single server from causing the directory service to become unavailable. At a minimum, replicate the local directory tree to at least one backup server.

Some directory architects argue that information should be replicated three times per physical location for maximum data reliability. The extent to use replication for fault tolerance depends on the environment and personal preferences, but base this decision on the quality of the hardware and networks used by the directory service. Unreliable hardware requires more backup servers.

NOTE

Do *not* use replication as a replacement for a regular data backup policy. For information on backing up the directory data, see the *Red Hat Directory Server Administrator's Guide* .

To guarantee write-failover for all directory clients, use a multi-master replication scenario. If read-failover is sufficient, use single-master replication.

LDAP client applications can usually be configured to search only one LDAP server. Unless there is a custom client application to rotate through LDAP servers located at different DNS host names, the LDAP client applications can only be configured to look up a single DNS host name for a Directory Server. Therefore, it is probably necessary to use either DNS round-robins or network sorts to provide failover to the backup Directory Servers. For information on setting up and using DNS round-robins or network sorts, see the DNS documentation.

7.3.7. Using Replication for Local Availability

The necessity of replicating for local availability is determined by the quality of the network as well as the activities of the site. In addition, carefully consider the nature of the data contained in the directory service and the consequences to the enterprise if that data were to become temporarily unavailable. The more mission-critical the data, the less tolerant the system is of outages caused by poor network connections.

Use replication for local availability for the following reasons:

- To keep a local master copy of the data.

  This is an important strategy for large, multinational enterprises that need to maintain directory information of interest only to the employees in a specific country. Having a local master copy of the data is also important to any enterprise where interoffice politics dictate that data be controlled at a divisional or organizational level.

- To mitigate unreliable or intermittently available network connections.

  Intermittent network connections can occur if there are unreliable WANs, as often occurs in international networks.

- To offset periodic, extremely heavy network loads that may cause the performance of the directory service to be severely reduced.

  Performance may also be affected in enterprises with aging networks, which may experience these conditions during normal business hours.

7.3.8. Using Replication for Load Balancing

Replication can balance the load on the Directory Servers in several ways:

- By spreading the users' search activities across several servers.

- By dedicating servers to read-only activities (writes occur only on the supplier server).

- By dedicating special servers to specific tasks, such as supporting mail server activities.

Balancing the workload of the network is an important function performed by directory data replication. Whenever possible, move data to servers that can be accessed using a reasonably fast and reliable network connection. The

most important considerations are the speed and reliability of the network connection between the server and the directory users.

Directory entries generally average around one kilobyte in size. Therefore, every directory lookup adds about one kilobyte to the network load. If the directory users perform ten directory lookups per day, then, for every directory user, there is an increased network load of around 10 kilobyte per day. If the site has a slow, heavily loaded, or unreliable WAN, then consider replica tinge the directory tree to a local server.

Also consider whether the benefit of locally available data is worth the cost of the increased network load caused by replication. If an entire directory tree is replicated to a remote site, for instance, that potentially adds a large strain on the network in comparison to the traffic caused by the users' directory lookups. This is especially true if the directory tree is changing frequently, yet there are only a few users at the remote site performing a few directory lookups per day.

Table 7.1, "Effects of Replication and Remote Lookup on the Network" compares the approximate cost of replicating a directory of one million entries, where 10% of those entries undergo daily change, with the cost of having a small remote site of 100 employees perform 10 lookups per day. In each case the average size of a directory entry is assumed to be 1Kb.

Table 7.1. Effects of Replication and Remote Lookup on the Network

Load Type	Objects[a]	Accesses/Day[b]	Avg. Entry Size	Load
Replication	1 million	100,000	1Kb	100Mb/day
Remote Lookup	100	1,000	1Kb	1Mb/day

[a] For replication, *objects* refers to the number of entries in the database. For remote lookup, it refers to the number of users who access the database.

[b] For replication, *Accesses/Day* is based on a 10% change rate to the database that needs to be replicated. For remote lookup, it is based on ten lookups per day for each remote user.

Given the difference in loads caused by replication versus that caused by normal directory usage, using replication for network load-balancing purposes may not be desirable. On the other hand, the benefits of locally available directory data can far outweigh any considerations regarding network loads.

A good compromise between making data available to local sites and overloading the network is to use scheduled replication. For more information on data consistency and replication schedules, see Section 7.1.2, "Data Consistency".

7.3.8.1. Example of Network Load Balancing

In this example, the enterprise has offices in New York and Los Angeles, and each office has specific subtrees that they manage.

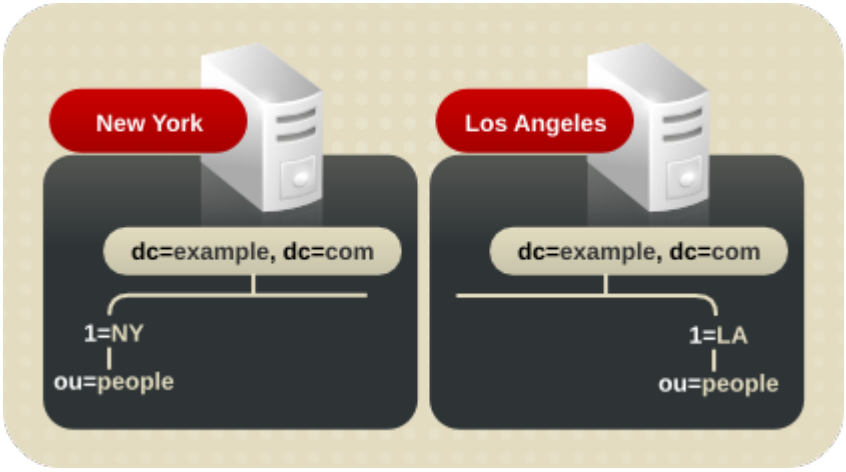


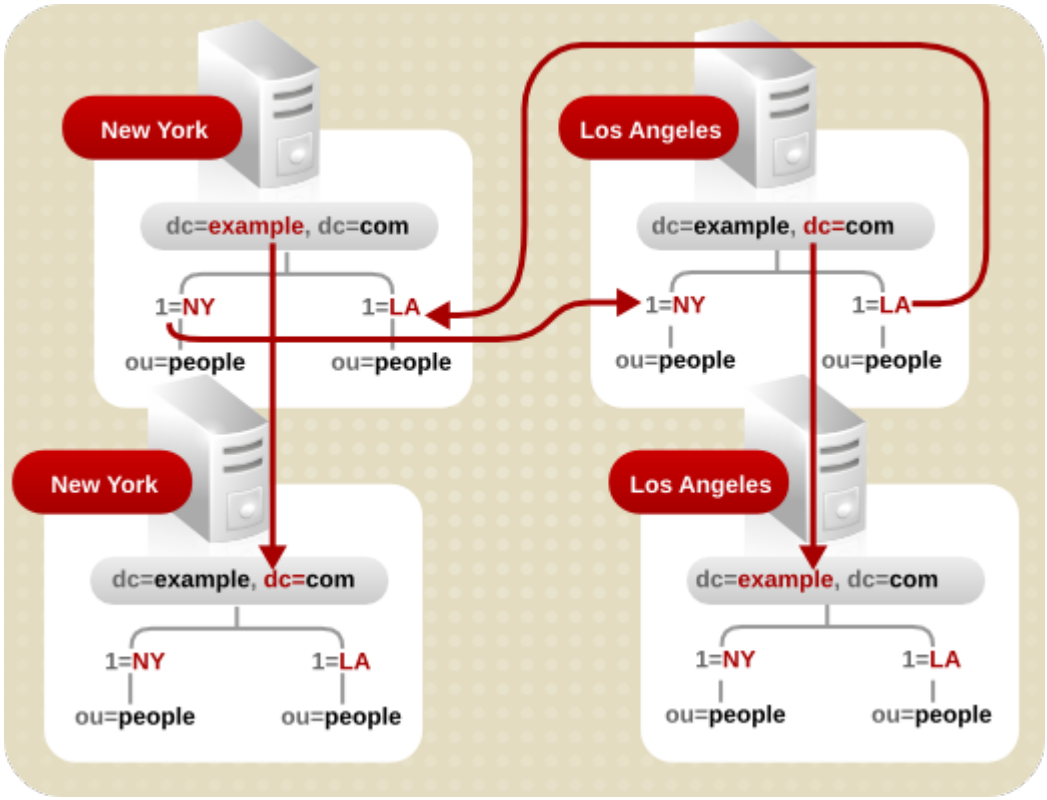
Figure 7.9. Managing Enterprise Subtrees in Remote Offices

Each office contains a high-speed network, but the connection between two cities is unreliable. To balance the network load:

1. Select one server in each office to be the supplier server for the locally managed data.

2. Replicate locally managed data from that server to the corresponding supplier server in the remote office.

3. Replicate the directory tree on each supplier server (including data supplied from the remote office) to at least one local Directory Server to ensure availability of the directory data. Use multi-master replication for the suffix that is managed locally, and cascading replication for the suffix that receives a master copy of the data from a remote server.



7.3.8.2. Example of Load Balancing for Improved Performance

Suppose that the enterprise has the following characteristics:

- Uses a Directory Server that includes 1.5 million entries in support of one million users
- Each user performs ten directory lookups per day
- Uses a messaging server that handles 25 million mail messages per day
- The messaging server performs five directory lookups for every mail message that it handles

This equates to ten million directory lookups per day for users, and 125 million directory lookups per day for email; a total of 135 million directory lookups per day.

With an eight-hour business day and users spread across four time zones, for example, the business day (or peak usage) across four time zones extends to 12 hours. Therefore, the service must support 135 million directory lookups in a 12-hour day. This equates to 3,125 lookups per second (135,000,000 / (60*60*12)).

Table 7.2. Calculating Directory Server Load

Access Type	Type Count	Accesses per Day	Total Accesses
User Lookup	1 million	10	10 million
Email Lookup	25 million	5	125 million
Combined accesses			135 million
Total		135 million (3,125/second)	

If the hardware that runs the Directory Servers supports 500 reads per second, at least six or seven Directory Servers must be used to support this load. For enterprises with a million directory users, add more Directory Servers for local availability purposes.

There are several different methods of replication:

- Place two Directory Servers in a multi-master configuration in one city to handle all write traffic.

  This configuration assumes that there should be a single point of control for all directory data.

- Use these supplier servers to replicate to one or more hub suppliers.

  The read, search, and compare requests serviced by the directory service should be targeted at the consumer servers, thereby freeing the supplier servers to handle write requests.

- Use the hub supplier to replicate to local sites throughout the enterprise.

  Replicating to local sites helps balance the workload of the servers and the WANs, as well as ensuring high availability of directory data.

- At each site, replicate at least once to ensure high availability, at least for read operations.

- Use DNS sort to ensure that local users always find a local Directory Server they can use for directory lookups.

7.3.8.3. Example Replication Strategy for a Small Site

Example Corp. has the following characteristics:

- The entire enterprise is contained within a single building.
- The building has a very fast (100 MB per second) and lightly used network.
- The network is very stable, and the server hardware and OS platforms are reliable.
- A single server is capable of easily handling the site's load.

In this case, Example Corp. decides to replicate at least once to ensure availability in the event the primary server is shut down for maintenance or hardware upgrades. Also, set up a DNS round-robin to improve LDAP connection performance in the event that one of the Directory Servers becomes unavailable.

7.3.8.4. Example Replication Strategy for a Large Site

As Example Corp. has grown, it retains its previous characteristics (as in Section 7.3.8.3, “Example Replication Strategy for a Small Site”) with a few changes:

- The enterprise is contained within two separate buildings.
- There are slow connections between the buildings, and these connections are very busy during normal business hours.

As their network needs changes, then Example Corp.'s administrators adjust their replication strategy:

- Choose a single server in one of the two buildings to contain a master copy of the directory data.

  This server should be placed in the building that contains the largest number of people responsible for the master copy of the directory data. We shall see this building as Building A.

- Replicate at least once within Building A for high availability of directory data.

  Use a multi-master replication configuration to ensure write-failover.

- Create two replicas in the second building (Building B).
- If there is no need for close consistency between the supplier and consumer server, schedule replication so that it occurs only during off-peak hours.

7.4. USING REPLICATION WITH OTHER DIRECTORY SERVER FEATURES

Replication interacts with other Directory Server features to provide advanced replication features. The following sections describe feature interactions to better design the replication strategy.

7.4.1. Replication and Access Control

The directory service stores ACIs as attributes of entries. This means that the ACI is replicated together with other directory content. This is important because Directory Server evaluates ACIs locally.

For more information about designing access control for the directory, see Chapter 9, *Designing a Secure Directory*.

7.4.2. Replication and Directory Server Plug-ins

Replication works with most of the plug-ins delivered with Directory Server. There are some exceptions and limitations in the case of multi-master replication with the following plug-ins:

- Attribute Uniqueness Plug-in

  The Attribute Uniqueness Plug-in validate attribute values added to local entries to make sure that all values are unique. However, this checking is done directly on the server, not replicated from other suppliers. For example, Example Corp. requires that the ***mail*** attribute be unique, but two users are added with the same ***mail*** attribute to two different supplier servers at the same time. As long as there it no a naming conflict, then there is no replication conflict, but the ***mail*** attribute is not unique.

- Referential Integrity Plug-in

  Referential integrity works with multi-master replication, provided that this plug-in is enabled on only one supplier in the multi-master set. This ensures that referential integrity updates occur on only one of the supplier servers and propagated to the others.

NOTE

By default, these plug-ins are disabled, and they must be manually enabled.

7.4.3. Replication and Database Links

With chaining to distribute directory entries, the server containing the database link references a remote server that contains the actual data. In this environment, the database link itself cannot be replicated. However, the database that contains the actual data on the remote server *can* be replicated.

Do not use the replication process as a backup for database links. Database links must be backed up manually. For more information about chaining and entry distribution, see Chapter 6, *Designing the Directory Topology*.

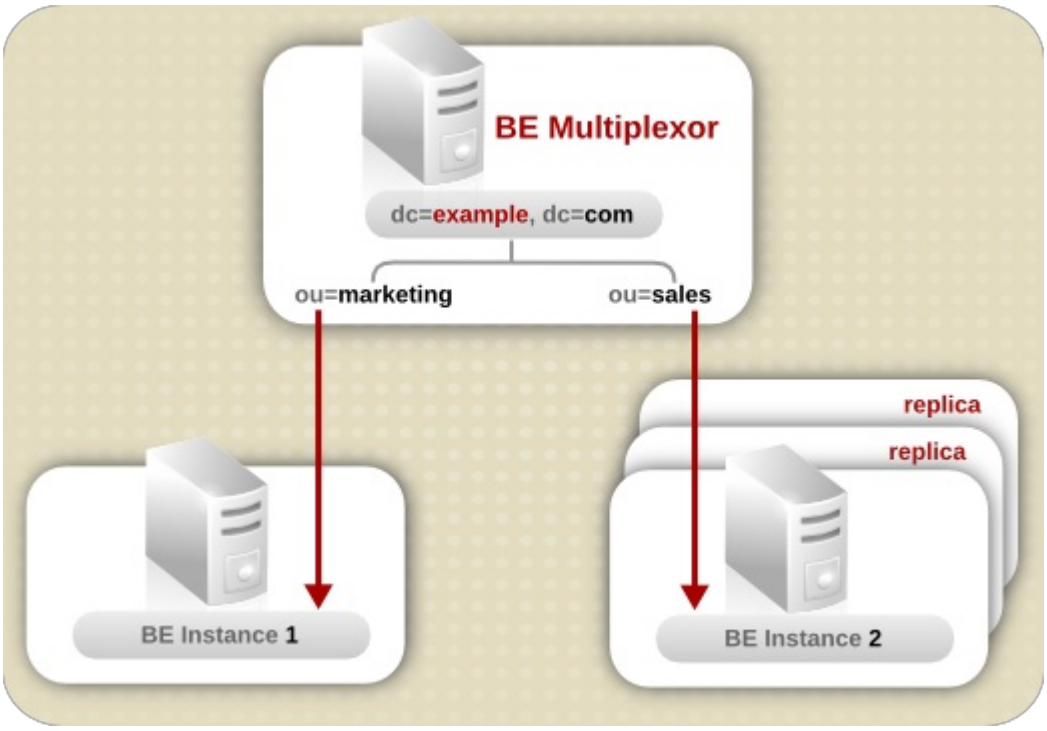


Figure 7.10. Replicating Chained Databases

7.4.4. Schema Replication

For the standard schema, before replicating data to consumer servers, the supplier server checks whether its own version of the schema is synchronized with the version of the schema stored on the consumer servers. The following conditions apply:

- If the schema entries on both supplier and consumers are the same, the replication operation proceeds.
- If the version of the schema on the supplier server is more recent than the version stored on the consumer, the supplier server replicates its schema to the consumer before proceeding with the data replication.
- If the version of the schema on the supplier server is older than the version stored on the consumer, the server may return many errors during replication because the schema on the consumer cannot support the new data.

NOTE

Schema replication still occurs, even if the schemas between the supplier and replica do not match.

Replicatable changes include changes to the schema made through the Directory Server console, changes made through ldapmodify, and changes made directly to the99user.ldif file. Custom schema files, and any changes made to custom schema files, are not replicated.

A consumer might contain replicated data from two suppliers, each with different schema. Whichever supplier was updated last wins, and its schema is propagated to the consumer.

WARNING

Never update the schema on a consumer server, because the supplier server is unable to resolve conflicts that occur, and replication fails. Schema should be maintained on a supplier server in a replicated topology.

The same Directory Server can hold read-write replicas for which it acts as a supplier and read-only replicas for which it acts as a consumer. Therefore, always identify the server that will function as a supplier for the schema, and then set up replication agreements between this supplier and all other servers in the replication environment that will function as consumers for the schema information.

Special replication agreements are not required to replicate the schema. If replication has been configured between a supplier and a consumer, schema replication occurs by default.

For more information on schema design, see Chapter 3, *Designing the Directory Schema*.

Custom Schema

If the standard **99user.ldif** file is used for custom schema, these changes are replicated to all consumers.

Custom schema files must be copied to each server in order to maintain the information in the same schema file on all servers. Custom schema files, and changes to those files, are not replicated, even if they are made through the Directory Server Console or **ldapmodify**.

If there are custom schema files, ensure that these files are copied to all servers after making changes on the supplier. After all of the files have been copied, restart the server.

For more information on custom schema files, see Section 3.4.7, “Creating Custom Schema Files”.

7.4.5. Replication and Synchronization

In order to propagate synchronized Windows entries throughout the Directory Server, use synchronization within a multi-master environment. Synchronization agreement should be kept to the lowest amount possible, preferably one per deployment. Multi-master replication allows the Windows information to be available throughout the network, while limiting the data access point to a single Directory Server.

CHAPTER 8. DESIGNING SYNCHRONIZATION

An important factor to consider while conducting the site survey for an existing site (Section 2.3, "Performing a Site Survey") is to include the structure and data types of Active Directory directory services. Through Windows Sync, an existing Windows directory service can be synchronized and integrated with the Directory Server, including creating, modifying, and deleting Windows accounts on the Directory Server or, oppositely, the Directory Server accounts on Windows. This provides an efficient and effective way to maintain directory information integrity across directory services.

8.1. WINDOWS SYNCHRONIZATION OVERVIEW

The synchronization process is analogous to the replication process: it is enabled by a plug-in and configured and initiated through a synchronization agreement, and a record of directory changes is maintained and updates are sent according to that log.

There are two parts to the complete Windows Synchronization process:

- *User and Group Sync.* As with multi-master replication, user and group entries are synchronized through a plug-in, which is enabled by default. The same changelog that is used for multi-master replication is also used to send updates from the Directory Server to the Windows synchronization peer server as an LDAP operation. The server also performs LDAP search operations against its Windows server to synchronize changes made to Windows entries to the corresponding Directory Server entry.

- *Password Sync.* This application captures password changes for Windows users and relays those changes back to the Directory Server over LDAPS. It must be installed on the Active Directory machine.

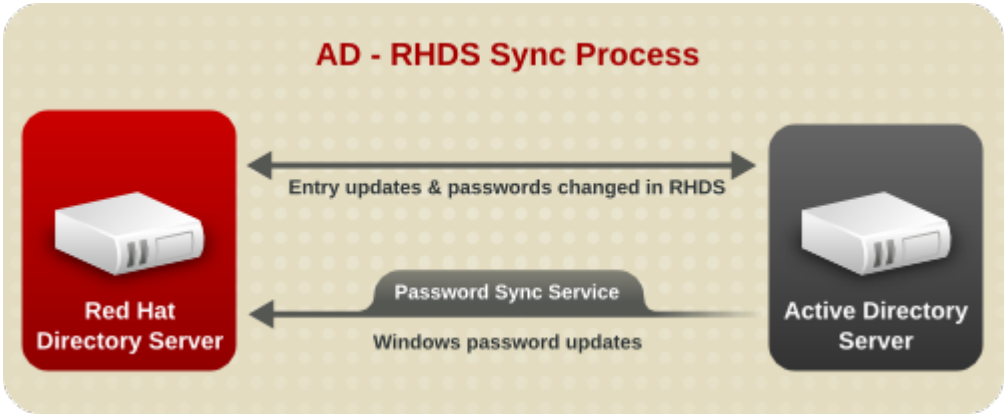


Figure 8.1. The Sync Process

8.1.1. Synchronization Agreements

Synchronization is configured and controlled by one or more *synchronization agreements* . These are similar in purpose to replication agreements and contain a similar set of information, including the host name and port number for the Windows server and the subtrees being synchronized. The Directory Server connects to its peer Windows server using LDAP or LDAP over SSL to both send and receive updates.

A single Windows subtree is synchronized with a single Directory Server subtree, and vice versa. Unlike replication, which connects *databases*, synchronization is between *suffixes*, parts of the directory tree structure. Therefore, when designing the directory tree, consider the Windows subtrees that should be synchronized with the Directory Server, and design or add corresponding Directory Server subtrees. The synchronized Windows and Directory Server suffixes are both specified in the synchronization agreement. All entries within the respective subtrees are available for synchronization, including entries that are not immediate children of the specified suffix.

NOTE

Any descendant container entries need to be created separately on the Windows server by an administrator; Windows Sync does not create container entries.

8.1.2. Changelogs

The Directory Server maintains a *changelog*, a database that records modifications that have occurred. The changelog is used by Windows Sync to coordinate and send changes made to the Windows synchronization peer server. Changes to entries in the Windows server are found by using Active Directory's Dirsync search feature. Because there is no changelog on the Active Directory side, the Dirsync search is issued, by default, periodically every five minutes. Using Dirsync ensures that only those entries that have changed since the previous search are retrieved.

8.2. SUPPORTED ACTIVE DIRECTORY VERSIONS

Windows Synchronization and the Password Sync Service are supported on Windows 2008 R2 on both 32-bit and 64-bit platforms.

8.3. PLANNING WINDOWS SYNCHRONIZATION

It may be useful to assess the type of information, Windows servers, and other considerations before setting up synchronization, similar to the site surveys for organizing data or planning replication.

8.3.1. Resource Requirements

Synchronization uses server resources. Consider the following resource requirements when defining the replication strategy:

- Disk usage — The changelog is written after each update operation. Servers receiving many update operations may see higher disk usage. In addition, a single changelog is maintained for all replication databases and synchronized databases. If a supplier contains multiple replicated and synchronized databases, the changelog is used more frequently, and the disk usage is even higher.
- Server threads — The synchronization agreement uses one server thread.
- File descriptors — The number of file descriptors available to the server is reduced by the changelog (one file descriptor) and each replication and synchronization agreement (one file descriptor per agreement).
- Quality of the LANs and WANs connecting different buildings or remote sites and the amount of available bandwidth.
- The number and size of the entries stored in the directory.

A site that manages human resource databases or financial information is likely to put a heavier load on the directory than a site containing engineering staff that uses the directory for simple telephone book purposes.

8.3.2. Managing Disk Space for the Changelog

As with multi-master replications, synchronization requires a changelog of to track directory edits and log entries for the state information for update entries, and tombstone entries for deleted entries. This information is required for synchronization. Because these log files can get very large, periodically cleaning up these files is necessary to keep from wasting disk space.

There are four attributes which can maintain the changelog. Two are under **cn=changelog5** and relate directly to trimming the changelog:

- ***nsslapd-changelogmaxage*** sets the maximum age that the entries in the changelog can be; once an entry is older than that limit, it is deleted. This keeps the changelog from growing indefinitely.
- ***nsslapd-changelogmaxentries*** sets the maximum number of entries that are allowed in the changelog. Like ***nsslapd-changelogmaxage***, this also trims the changelog, but be careful about the setting. This must be large enough to allow a complete set of directory information or synchronization may not function properly.

The other two attributes are under the synchronization agreement entry in **cn=***sync_agreement*, **cn=WindowsReplica,cn=***suffixDN***,cn=mapping tree,cn=config**. These two attributes relate to maintenance information kept in the changelog, the tombstone and state information, rather than the directory edits information.

- ***nsDS5ReplicaPurgeDelay*** sets the maximum age that tombstone (deleted) entries and state information can be in the changelog. Once a tombstone or state information entry is older than that age, it is deleted. This differs from the ***nsslapd-changelogmaxage*** attribute in that the ***nsDS5ReplicaPurgeDelay*** value applies only to tombstone and state information entries; ***nsslapd-changelogmaxage*** applies to every entry in the changelog, including directory modifications.
- ***nsDS5ReplicaTombstonePurgeInterval*** sets the frequency which the server runs a purge operation. At this interval, the Directory Server runs an internal operation to clean the tombstone and state entries out of the changelog. Make sure that the maximum age is longer than the longest replication update schedule or multi-master replication may not be able to update replicas properly.

The parameters for managing replication and the changelog are described in chapter 2, "Core Configuration Attributes," in the *Configuration, Command, and File Reference* .

8.3.3. Defining the Connection Type

Synchronization can occur using simple authentication over a standard port, using SSL/TLS, or using Start TLS (a secure connection over a standard port).

Although it is not required, it is strongly recommended that SSL or other secure connection be used for synchronization. If passwords are going to be synchronized from the Windows server, then SSL must be enabled on both servers so the synchronization proceeds over a secure port.

8.3.4. Considering a Data Master

The *data master* is the server that is the master source of data; this is the primary or authoritative source for data.

Windows and Directory Server services are kept continuously synchronized through the synchronization agreement, which minimizes potential conflicts between the two services. However, if the Directory Server is part of a replication deployment, then conflicts could arise between changes made within the Directory Server replication scenario and the Windows domain depending on the replication schedule.

Consider which server will be the data master when the data resides in two different directory services, and decide how much of that information will be shared. The best course is to choose a single directory service to master the data and allow the synchronization process to add, update, or delete the entries on the other service.

Choose one area (Windows domain or Directory Server) to master the data. Alternatively, choose a single Directory Server as a data master and synchronize it with each Windows domain. If the Directory Server is involved in replication, design the replication structure to avoid conflicts, losing data, or overwriting data.

How master copies of the data are maintained depends on the specific needs of the deployment. Regardless of how data masters are maintained, keep it simple and consistent. For example, do not attempt to master data in multiple sites, then automatically exchange data between competing applications. Doing so leads to a "last change wins" scenario and increases administrative overhead.

8.3.5. Determining the Subtree to Synchronize

Only a single Directory Server subtree can be synchronized to a single Windows subtree, and it is recommended that there only be a single synchronization agreement between directory services. Select or design the parts of the directory trees to synchronize; consider designing special suffixes specifically for synchronized entries.

The subtree plan should also account for entries which may correspond between the Active Directory and Directory Server directories but are out of the scope of the synced subtree. The synchronization process actually starts at the root DN to begin evaluating entries for synchronization. Entries are correlated based on the ***samAccount*** in the Active Directory and the ***uid*** attribute in Directory Server. The synchronization plug-in notes if an entry (based on the ***samAccount*/*uid*** relationship) is removed from the synced subtree either because it is deleted or moved. That is the signal to the synchronization plug-in that the entry is no longer to be synced. The issue is that the sync process needs some configuration to determine how to handle that moved entry. There are three options which can be set in the sycnrhonization agreement: delete the corresponding Directory Server entry, ignore the change (the default), or unsync the entries but leave them otherwise intact.

8.3.6. Interaction with a Replicated Environment

Synchronization links a Directory Server suffix and subtree (for example, ***ou=People,dc=example,dc=com***) to a corresponding Windows domain and subtree (***cn=Users,dc=test,dc=com***). Each subtree can be synchronized only to one other subtree to avoid naming conflicts and change conflicts.

To take advantage of Windows Sync, use it with a Directory Server supplier in multi-master replication synchronized to a member of a Windows domain. This propagates changes through both directory systems while keeping the information centralized and easy to maintain. It also makes it easier to master the data.

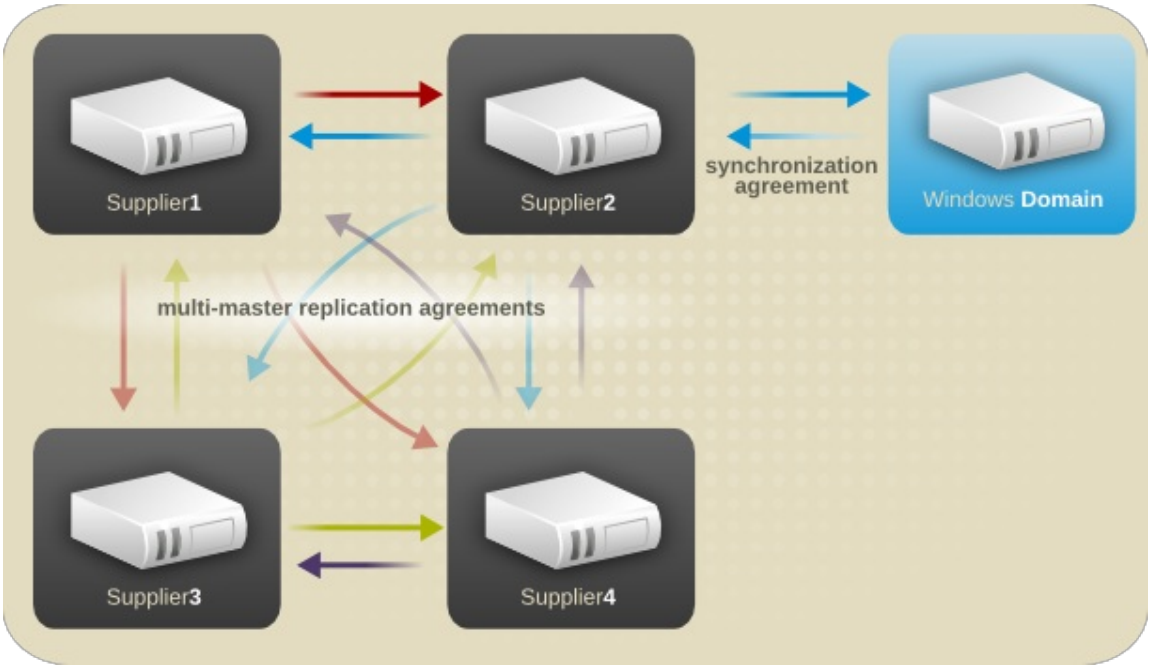


Figure 8.2. Multi-Master Directory Server — Windows Domain Synchronization

Only create one synchronization agreement to any given Windows domain. To propagate the changes and information synchronized from the Windows server throughout the Directory Server, create the synchronization agreement with a multi-master supplier, preferably a data master for the replication deployment.

8.3.7. Controlling the Sync Direction

As Figure 8.1, "The Sync Process" illustrates, synchronization is *bi-directional* by default. That means that changes in Active Directory are sent to Directory Server and changes on Directory Server are sent to Active Directory.

It is possible to create *uni-directional* synchronization by adding the ***oneWaySync*** parameter to the sync agreement. This attribute defines which direction to send changes.

To send changes *from* the Active Directory server *to* the Directory Server, the value is ***fromWindows***. In this case, during the regular synchronization update interval, the Directory Server contacts the Active Directory server and sends the DirSync control to request updates. However, the Directory Server does not send any changes or entries from its side. So, the sync update consists of the Active Directory changes being sent to and updating the Directory Server entries.

To sync changes *from* the Directory Server *to* the Active Directory server, the value is ***toWindows***. The Directory Server sends entry modifications to the Active Directory server in a normal update, but it does not include the DirSync control so that it does not request any updates from the Active Directory side.

Enabling uni-directional sync does *not* automatically prevent changes on the un-synchronized server, and this can lead to inconsistencies between the sync peers between sync updates. For example, uni-directional sync is configured to go from Active Directory to Directory Server, so Active Directory is (in essence) the data master. If an entry is modified or even deleted on the Directory Server, then the Directory Server information is different then the information and those changes are never carried over to Active Directory. During the next sync update, the edits are overwritten on the Directory Server and the deleted entry is re-added.

To prevent data inconsistency, use access control rules to prevent editing or deleting entries within the synchronized subtree on the *un*synced server. Access controls for Directory Server are covered in Section 9.7, "Designing Access Control". For Active Directory, see the appropriate Windows documentation.

8.3.8. Controlling Which Entries Are Synced

Windows Sync provides some control over which entries are synchronized to give sufficient flexibility to support different deployment scenarios. This control is set through different configuration attributes set in the Directory Server:

- Within the Windows subtree, only user and group entries can be synchronized to Directory Server. When creating the synchronization agreement, there is an option to synchronize new Windows user and group entries as they are created. If these attributes are set to **on**, then existing Windows entries are synchronized to the Directory Server, and entries as they are created in the Windows server are synchronized to the Directory Server.

- As with Active Directory entries, only user and group entries in Directory Server can be synchronized. Entries which will be synchronized must have the **ntUser** or **ntGroup** object classes and required attributes; all other entries are ignored.

Directory Server passwords are synchronized along with other entry attributes because plaintext passwords are retained in the Directory Server changelog. The Password Sync Service is needed to catch password changes made on the Windows server. Without the Password Sync Service, it would be impossible to have Windows passwords synchronized because passwords are hashed in the Windows server, and the Windows hashing function is incompatible with the one used by Directory Server.

8.3.9. Identifying the Directory Data to Synchronize

Windows Sync synchronizes user and group entries between directory services. After deciding which subtrees to synchronize, plan the information to store in those subtrees, such as the following:

- Contact information for directory users and employees, such as telephone numbers, home and office addresses, and email addresses.
- Contact information for trading partners, clients, and customers.
- User's software preferences or software configuration information.
- Group information and group membership.

  Group members are synchronized only if they are within the synchronized suffix. Group members that are not within the scope of the agreement are left unchanged on both sides; that is, they are listed as members of the group on the appropriate directory service, but their ***member*** attribute in the group entry is not synchronized with the synchronization peer.

Which entries are synchronized is set in the synchronization agreement. User entries are synchronized separately from group entries. Additionally, deleting entries is configured separately; deletions have to be specifically synchronized.

In the Directory Server, only entries that contain the **ntGroup** or **ntUser** object classes and required attributes are synchronized; determine what existing and future entries should be synchronized with the Windows server.

After determining what entries should be present in the directory, determine what attributes of these objects need to be maintained in the directory. Only a subset of the possible attributes for Directory Server or for Active Directory are synchronized. Additionally, this subset of attributes can be limited even more by excluding certain attributes through the sync agreement (fractional synchronization).

Plan both the entries and the data contained in those entries according to the available synchronization attributes. The synchronized attributes and the differences between Directory Server and Active Directory schema are described in Section 8.4, “Schema Elements Synchronized Between Active Directory and Directory Server” .

8.3.10. Synchronizing POSIX Attributes for Users and Groups

A subset of all possible user and attributes are synchronized between Active Directory and Red Hat Directory Server. Some attributes are mapped, where there are differences between Active Directory and Directory Server schemas, and some attributes are matched directly. By default, only those attributes are synchronized.

One type of attribute that is missing from that sync list is any POSIX-related attribute. On Linux systems, system users and groups are identified as POSIX entries, and LDAP POSIX attributes contain that required information. However, when Windows users are synced over, they have ***ntUser*** and ***ntGroup*** attributes automatically added which identify them as Windows accounts, but no POSIX attributes are synced over (even if they exist on the Active Directory entry) and no POSIX attributes are added on the Directory Server side.

The Posix Winsync API Plug-in synchronizes POSIX attributes between Active Directory and Directory Server entries. This plug-in is disabled by default, but when enabled, it allows identifying POSIX attributes to be set in the data master and then synchronized over to the peer server. This is beneficial if Active Directory is used as the user account store since the POSIX attributes can be set directly in the Active Directory entries and then synced and preserved over in the Directory Server directory.

NOTE

All POSIX attributes (such as ***uidNumber***, ***gidNumber***, and ***homeDirectory***) are synchronized between Active Directory and Directory Server entries if the plug-in is enabled. However, if a new POSIX entry or POSIX attributes are added to an existing entry in the Directory Server, *only the POSIX attributes are synchronized over to the Active Directory corresponding entry*. The POSIX object class (**posixAccount** for users and **posixGroup** for groups) is not added to the Active Directory entry.

8.3.11. Synchronizing Passwords and Installing Password Services

While the DirSync plug-in is installed with the Directory Server and enabled by default, an additional Windows service, Password Sync, must be installed on the Windows machine to synchronize passwords. This service is required to transfer any password changes made on the Windows server over to the Directory Server.

Unless the Password Sync service is installed, password synchronization (synchronizing the ***userPassword*** attribute) is not enabled. What this means is that even if Directory Server user entries are synchronized over to the Windows server, the user entries are not active on the Windows domain (meaning, among other things, those synced users cannot log into the domain, since they do not have a password).

NOTE

There is a password policy attribute called ***passwordTrackUpdateTime*** which records a separate timestamp for the last update of the user password. This can make it simpler to synchronize password changes between Active Directory and Directory Server or with other clients.

8.3.12. Defining an Update Strategy

Existing Directory Server entries that are modified to contain the necessary synchronization attributes are not synchronized until the next total update. Modifications to Windows entries and Directory Server entries that have already been synchronized are carried at the next incremental update. As a part of this strategy, try to master data in a single place, limiting the applications that can change the data, and schedule necessary total updates (these updates do not overwrite or delete existing information; they add new entries and send modifications).

By default, the Windows and Directory Server instances are kept constantly in sync and have changes published every five minutes. This schedule can be altered by manually setting the sync agreement attributes to change the update interval (***winSyncInterval***) or by setting a different update schedule (***nsDS5ReplicaUpdateSchedule***).

8.3.13. Editing the Sync Agreement

The basic sync agreement configured through the Directory Server Console sets very simple information about synchronization, like the host and port information, synchronized subtrees, and connection types.

However, many configurations available to multi-master replication, like fractional replication and sync schedules, are available to Windows-Directory Server synchronization. These settings must simply be added to the sync agreement manually.

Changing the default sync agreement is described in the *Administrator's Guide*, and the available sync agreement attributes are listed in the *Configuration, Command, and File Reference* .

8.4. SCHEMA ELEMENTS SYNCHRONIZED BETWEEN ACTIVE DIRECTORY AND DIRECTORY SERVER

All synchronized entries in the Directory Server, whether they originated in the Directory Server or in the Windows server, have the following special synchronization attributes:

- ***ntUniqueId*** contains the value of the ***objectGUID*** attribute for the corresponding Windows entry. This attribute is set by the synchronization process and should not be set or modified manually.
- ***ntUserDeleteAccount*** is set automatically when a Windows entry is synchronized but must be set manually for Directory Server entries. If ***ntUserDeleteAccount*** has the value **true**, the corresponding Windows entry is deleted when the Directory Server entry is deleted.
- ***ntDomainUser*** corresponds to the ***samAccountName*** attribute for Active Directory entries. *User entries only.*
- ***ntGroupType*** is set automatically for Windows groups that are synchronized, but must be set manually on Directory Server entries before they are synchronized. *Group entries only.*

A pre-defined list of attributes are synchronized between Directory Server and Active Directory entries. Some of these attributes are the same, like the ***givenName*** attribute in Directory Server matches the ***givenName*** attribute in Active Directory. Because the defined schema in Active Directory and Red Hat Directory Server are slightly different, other attributes are mapped between Active Directory and Red Hat Directory Server; most of these are Windows-specific attributes in Directory Server.

8.4.1. User Attributes Synchronized Between Directory Server and Active Directory

Only a subset of Directory Server and Active Directory attributes are synchronized. These attributes are hardcoded and are defined regardless of which way the entry is being synchronized. Any other attributes present in the entry, either in Directory Server or in Active Directory, remain unaffected by synchronization.

Some attributes used in Directory Server and Active Directory are identical. These are usually attributes defined in an LDAP standard, which are common among all LDAP services. These attributes are synchronized to one another exactly. Table 8.2, "User Schema That Are the Same in Directory Server and Windows Servers" shows attributes that are the same between the Directory Server and Windows servers.

Some attributes define the same information, but the names of the attributes or their schema definitions are different. These attributes are mapped between Active Directory and Directory Server, so that attribute A in one server is treated as attribute B in the other. For synchronization, many of these attributes relate to Windows-specific information. Table 8.1, "User Schema Mapped between Directory Server and Active Directory" shows the attributes that are mapped between the Directory Server and Windows servers.

For more information on the differences in ways that Directory Server and Active Directory handle some schema elements, see Section 8.4.2, "User Schema Differences between Red Hat Directory Server and Active Directory" .

Table 8.1. User Schema Mapped between Directory Server and Active Directory

Directory Server	Active Directory
cn	name
ntUserDomainId	sAMAccountName
ntUserHomeDir	homeDirectory
ntUserScriptPath	scriptPath
ntUserLastLogon	lastLogon
ntUserLastLogoff	lastLogoff
ntUserAcctExpires	accountExpires
ntUserCodePage	codePage
ntUserLogonHours	logonHours
ntUserMaxStorage	maxStorage
ntUserProfile	profilePath
ntUserParms	userParameters
ntUserWorkstations	userWorkstations

Table 8.2. User Schema That Are the Same in Directory Server and Windows Servers

cn	physicalDeliveryOfficeName
description	postOfficeBox
destinationIndicator	postalAddress
facsimileTelephoneNumber	postalCode
givenName	registeredAddress
homePhone	sn
homePostalAddress	st

initials	street
l	telephoneNumber
mail	teletexTerminalIdentifier
manager	telexNumber
mobile	title
o	userCertificate
ou	x121Address
pager	

8.4.2. User Schema Differences between Red Hat Directory Server and Active Directory

Although Active Directory supports the same basic X.500 object classes as Directory Server, there are a few incompatibilities of which administrators should be aware.

8.4.2.1. Values for cn Attributes

In Directory Server, the ***cn*** attribute can be multi-valued, while in Active Directory this attribute must have only a single value. When the Directory Server ***cn*** attribute is synchronized, then, only one value is sent to the Active Directory peer.

What this means for synchronization is that,potentially, if a ***cn*** value is added to an Active Directory entry and that value is not one of the values for ***cn*** in Directory Server, then all of the Directory Server ***cn*** values are overwritten with the single Active Directory value.

One other important difference is that Active Directory uses the ***cn*** attribute attribute as its naming attribute, where Directory Server uses ***uid***. This means that there is the potential to rename the entry entirely if the ***cn*** attribute is edited in the Directory Server. If that ***cn*** change is written over to the Active Directory entry, then the entry is renamed, and the new named entry is written back over to Directory Server. This only happens, however, if the ***cn*** attribute is synced. If the change is not synchronized, then the entry is not renamed.

8.4.2.2. Password Policies

Both Active Directory and Directory Server can enforce password policies such as password minimum length or maximum age. Windows Sync makes no attempt to ensure that the policies are consistent, enforced, or synchronized. If password policy is not consistent in both Directory Server and Active Directory, then password changes made on one system may fail when synched to the other system. The default password syntax setting on Directory Server mimics the default password complexity rules that Active Directory enforces.

8.4.2.3. Values for street and streetAddress

Active Directory uses the attribute ***streetAddress*** for a user or group's postal address; this is the way that Directory Server uses the ***street*** attribute. There are two important differences in the way that Active Directory and Directory Server use the ***streetAddress*** and ***street*** attributes, respectively:

- In Directory Server, ***streetAddress*** is an alias for ***street***. Active Directory also has the ***street*** attribute, but it is a separate attribute that can hold an independent value, not an alias for ***streetAddress***.
- Active Directory defines both ***streetAddress*** and ***street*** as single-valued attributes, while Directory Server defines ***street*** as a multi-valued attribute, as specified in RFC 4519.

Because of the different ways that Directory Server and Active Directory handle ***streetAddress*** and ***street*** attributes, there are two rules to follow when setting address attributes in Active Directory and Directory Server:

- Windows Sync maps ***streetAddress*** in the Windows entry to ***street*** in Directory Server. To avoid conflicts, the ***street*** attribute should not be used in Active Directory.
- Only one Directory Server ***street*** attribute value is synced to Active Directory. If the ***streetAddress*** attribute is changed in Active Directory and the new value does not already exist in Directory Server, then all ***street*** attribute values in Directory Server are replaced with the new, single Active Directory value.

8.4.2.4. Constraints on the initials Attribute

For the ***initials*** attribute, Active Directory imposes a maximum length constraint of six characters, but Directory Server does not have a length limit. If an ***initials*** attribute longer than six characters is added to Directory Server, the value is trimmed when it is synchronized with the Active Directory entry.

8.4.3. Group Attributes Synchronized Between Directory Server and Active Directory

Only a subset of Directory Server and Active Directory attributes are synchronized. These attributes are hardcoded and are defined regardless of which way the entry is being synchronized. Any other attributes present in the entry, either in Directory Server or in Active Directory, remain unaffected by synchronization.

Some attributes used in Directory Server and Active Directory group entries are identical. These are usually attributes defined in an LDAP standard, which are common among all LDAP services. These attributes are synchronized to one another exactly. Table 8.4, "Group Entry Attributes That Are the Same between Directory Server and Active Directory" shows attributes that are the same between the Directory Server and Windows servers.

Some attributes define the same information, but the names of the attributes or their schema definitions are different. These attributes are mapped between Active Directory and Directory Server, so that attribute A in one server is treated as attribute B in the other. For synchronization, many of these attributes relate to Windows-specific information. Table 8.3, "Group Entry Attribute Mapping between Directory Server and Active Directory" shows the attributes that are mapped between the Directory Server and Windows servers.

For more information on the differences in ways that Directory Server and Active Directory handle some schema elements, see Section 8.4.4, "Group Schema Differences between Red Hat Directory Server and Active Directory" .

Table 8.3. Group Entry Attribute Mapping between Directory Server and Active Directory

Directory Server	Active Directory
cn	name
ntGroupAttributes	groupAttributes
ntGroupId	cn name sAMAccountName
ntGroupType	groupType

Table 8.4. Group Entry Attributes That Are the Same between Directory Server and Active Directory

cn	member
description	ou
l	seeAlso

8.4.4. Group Schema Differences between Red Hat Directory Server and Active Directory

Although Active Directory supports the same basic X.500 object classes as Directory Server, there are a few incompatibilities of which administrators should be aware.

Nested groups (where a group contains another group as a member) are supported and for WinSync are synchronized. However, Active Directory imposes certain constraints as to the composition of nested groups. For example, a global group contain a domain local group as a member. Directory Server has no concept of local and global groups, and, therefore, it is possible to create entries on the Directory Server side that violate Active Directory's constraints when synchronized.

CHAPTER 9. DESIGNING A SECURE DIRECTORY

How the data in Red Hat Directory Server are secured affects all of the previous design areas. Any security design needs to protect the data contained by the directory and meet the security and privacy needs of the users and applications.

This chapter describes how to analyze the security needs and explains how to design the directory to meet these needs.

9.1. ABOUT SECURITY THREATS

There are many potential threats to the security of the directory. Understanding the most common threats helps outline the overall security design. Threats to directory security fall into three main categories:

- Unauthorized access
- Unauthorized tampering
- Denial of service

9.1.1. Unauthorized Access

Protecting the directory from unauthorized access may seem straightforward, but implementing a secure solution may be more complex than it first appears. A number of potential access points exist on the directory information delivery path where an unauthorized client may gain access to data.

For example, an unauthorized client can use another client's credentials to access the data. This is particularly likely when the directory uses unprotected passwords. An unauthorized client can also eavesdrop on the information exchanged between a legitimate client and Directory Server.

Unauthorized access can occur from inside the company or, if the company is connected to an extranet or to the Internet, from outside the company.

The following scenarios describe just a few examples of how an unauthorized client might access the directory data.

The authentication methods, password policies, and access control mechanisms provided by the Directory Server offer efficient ways of preventing unauthorized access. See the following sections for more information:

- Section 9.4, “Selecting Appropriate Authentication Methods”
- Section 9.6, “Designing a Password Policy”
- Section 9.7, “Designing Access Control”

9.1.2. Unauthorized Tampering

If intruders gain access to the directory or intercept communications between Directory Server and a client application, they have the potential to modify (or tamper with) the directory data. The directory service is useless if the data can no longer be trusted by clients or if the directory itself cannot trust the modifications and queries it receives from clients.

For example, if the directory cannot detect tampering, an attacker could change a client's request to the server (or not forward it) and change the server's response to the client. SSL and similar technologies can solve this problem by signing information at either end of the connection. For more information about using SSL with Directory Server, see Section 9.9, “Securing Server Connections”.

9.1.3. Denial of Service

In a denial of service attack, the attacker's goal is to prevent the directory from providing service to its clients. For example, an attacker might use all of the system's resources, thereby preventing these resources from being used by anyone else.

Directory Server can prevent denial of service attacks by setting limits on the resources allocated to a particular bind DN. For more information about setting resource limits based on the user's bind DN, see the "User Account Management" chapter in the *Red Hat Directory Server Administrator's Guide* .

9.2. ANALYZING SECURITY NEEDS

Analyze the environment and users to identify specific security needs. The site survey in Chapter 3, *Designing the Directory Schema* clarifies some basic decisions about who can read and write the individual pieces of data in the directory. This information forms the basis of the security design.

The way security is implemented also depends on how the directory service is used to support the business. A directory that serves an intranet does not require the same security measures as a directory that supports an extranet or e-commerce applications that are open to the Internet.

If the directory only serves an intranet, consider what level of access is needed for information:

- How to provide users and applications with access to the information they need to perform their jobs.
- How to protect sensitive data regarding employees or the business from general access.

If the directory serves an extranet or supports e-commerce applications over the Internet, there are additional points to consider:

- How to offer customers a guarantee of privacy.
- How to guarantee information integrity.

The following sections provide information about analyzing security needs.

9.2.1. Determining Access Rights

The data analysis identifies what information users, groups, partners, customers, and applications need to access the directory service.

Access rights can be granted in one of two ways:

- Grant all categories of users as many rights as possible while still protecting sensitive data.

  An open method requires accurately determining what data are sensitive or critical to the business.

- Grant each category of users the minimum access they require to do their jobs.

  A restrictive method requires minutely understanding the information needs of each category of user inside, and possibly outside, of the organization.

Irrespective of the method used to determine access rights, create a simple table that lists the categories of users in the organization and the access rights granted to each. Consider creating a table that lists the sensitive data held in the directory and, for each piece of data, the steps taken to protect it.

For information about checking the identity of users, see Section 9.4, "Selecting Appropriate Authentication Methods". For information about restricting access to directory information, see Section 9.7, "Designing Access Control"

9.2.2. Ensuring Data Privacy and Integrity

When using the directory to support exchanges with business partners over an extranet or to support e-commerce applications with customers on the Internet, ensure the privacy and the integrity of the data exchanged.

There are several ways to do this:

- By encrypting data transfers.
- By using certificates to sign data transfers.

For information about encryption methods provided in Directory Server, see Section 9.6.2.11, "Password Storage Schemes"

For information about signing data, see Section 9.9, "Securing Server Connections" .

For information about encrypting sensitive information as it is stored in the Directory Server database, see Section 9.8, "Encrypting the Database"

9.2.3. Conducting Regular Audits

As an extra security measure, conduct regular audits to verify the efficiency of the overall security policy by examining the log files and the information recorded by the SNMP agents.

For more information about SNMP, see the *Red Hat Directory Server Administrator's Guide* . For more information about log files and SNMP, see the *Red Hat Directory Server Administrator's Guide* .

9.2.4. Example Security Needs Analysis

The examples provided in this section illustrate how the imaginary ISP company "example.com" analyzes its security needs.

example.com's business is to offer web hosting and Internet access. Part of example.com's activity is to host the directories of client companies. It also provides Internet access to a number of individual subscribers.

Therefore, example.com has three main categories of information in its directory:

- example.com internal information
- Information belonging to corporate customers
- Information pertaining to individual subscribers

example.com needs the following access controls:

- Provide access to the directory administrators of hosted companies (example_a and example_b) to their own directory information.
- Implement access control policies for hosted companies' directory information.
- Implement a standard access control policy for all individual clients who use example.com for Internet access from their homes.
- Deny access to example.com's corporate directory to all outsiders.
- Grant read access to example.com's directory of subscribers to the world.

9.3. OVERVIEW OF SECURITY METHODS

Directory Server offers several methods to design an overall security policy that is adapted to specific needs. The security policy should be strong enough to prevent sensitive information from being modified or retrieved by unauthorized users, but also simple enough to administer easily. A complex security policy can lead to mistakes that either prevent people from accessing information that they need to access or, worse, allow people to modify or retrieve directory information that they should not be allowed to access.

Table 9.1. Security Methods Available in Directory Server

Security Method	Description
Authentication	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.
Password policies	Defines the criteria that a password must satisfy to be considered valid; for example, age, length, and syntax.
Encryption	Protects the privacy of information. When data is encrypted, it is scrambled in a way that only the recipient can understand.
Access control	Tailors the access rights granted to different directory users and provides a means of specifying required credentials or bind attributes.
Account deactivation	Disables a user account, group of accounts, or an entire domain so that all authentication attempts are automatically rejected.
Secure connections	Maintains the integrity of information by encrypting connections with SSL, Start TLS, or SASL. If information is encrypted during transmission, the recipient can determine that it was not modified during transit. Secure connections can be required by setting a minimum security strength factor.

Security Method	Description
Auditing	Determines if the security of the directory has been compromised; on simple auditing method is reviewing the log files maintained by the directory.
SELinux	Uses security policies on the Red Hat Enterprise Linux machine to restrict and control access to Directory Server files and processes.

Combine any number of these tools for maintaining security in the security design, and incorporate other features of the directory service, such as replication and data distribution, to support the security design.

9.4. SELECTING APPROPRIATE AUTHENTICATION METHODS

A basic decision regarding the security policy is how users access the directory. Are anonymous users allowed to access the directory, or is every user required to log into the directory with a user name and password (authenticate)?

Directory Server provides the following methods for authentication:

- Section 9.4.1, "Anonymous and Unauthenticated Access"
- Section 9.4.2, "Simple Binds and Secure Binds"
- Section 9.4.3, "Certificate-Based Authentication"
- Section 9.4.4, "Proxy Authentication"
- Section 9.4.6, "Password-less Authentication"

The directory uses the same authentication mechanism for all users, whether they are people or LDAP-aware applications.

For information about preventing authentication by a client or group of clients, see Section 9.5, "Designing an Account Lockout Policy".

9.4.1. Anonymous and Unauthenticated Access

Anonymous access provides the easiest form of access to the directory. It makes data available to any user of the directory, regardless of whether they have authenticated.

However, anonymous access does not allow administrators to track who is performing what kinds of searches, only that someone is performing searches. With anonymous access, anyone who connects to the directory can access the data.

Therefore, an administrator may attempt to block a specific user or group of users from accessing some kinds of directory data, but, if anonymous access is allowed to that data, those users can still access the data simply by binding to the directory anonymously.

Anonymous access can be limited. Usually directory administrators only allow anonymous access for read, search, and compare privileges (not for write, add, delete, or selfwrite). Often, administrators limit access to a subset of attributes that contain general information such as names, telephone numbers, and email addresses. Anonymous access should never be allowed for more sensitive data such as government identification numbers (for example, Social Security Numbers in the US), home telephone numbers and addresses, and salary information.

Anonymous access can also be disabled entirely, if there is a need for tighter rules on who accesses the directory data.

An *unauthenticated bind* is when a user attempts to bind with a user name but without a user password attribute. For example:

```
ldapsearch -x -D "cn=jsmith,ou=people,dc=example,dc=com" -b "dc=example,dc=com" "(cn=joe)"
```

The Directory Server grants anonymous access if the user does not attempt to provide a password. An unauthenticated bind does not require that the bind DN be an existing entry.

As with anonymous binds, unauthenticated binds can be disabled to increase security by limiting access to the database. Disabling unauthenticated binds has another advantage: it can be used to prevent silent bind failures for

clients. A poorly-written application may believe that it successfully authenticated to the directory because it received a bind success message when, in reality, it failed to pass a password and simply connected with an unauthenticated bind.

9.4.2. Simple Binds and Secure Binds

If anonymous access is not allowed, users must authenticate to the directory before they can access the directory contents. With simple password authentication, a client authenticates to the server by sending a reusable password.

For example, a client authenticates to the directory using a bind operation in which it provides a distinguished name and a set of credentials. The server locates the entry in the directory that corresponds to the client DN and checks whether the password given by the client matches the value stored with the entry. If it does, the server authenticates the client. If it does not, the authentication operation fails, and the client receives an error message.

The bind DN often corresponds to the entry of a person. However, some directory administrators find it useful to bind as an organizational entry rather than as a person. The directory requires the entry used to bind to be of an object class that allows the ***userPassword*** attribute. This ensures that the directory recognizes the bind DN and password.

Most LDAP clients hide the bind DN from the user because users may find the long strings of DN characters hard to remember. When a client attempts to hide the bind DN from the user, it uses a bind algorithm such as the following:

1. The user enters a unique identifier, such as a user ID (for example, **fchen**).
2. The LDAP client application searches the directory for that identifier and returns the associated distinguished name (such as **uid=fchen,ou=people,dc=example,dc=com**).
3. The LDAP client application binds to the directory using the retrieved distinguished name and the password supplied by the user.

Simple password authentication offers an easy way to authenticate users, but it requires extra security to be used safely. Consider restricting its use to the organization's intranet. To use with connections between business partners over an extranet or for transmissions with customers on the Internet, it may be best to require a secure (encrypted) connection.

NOTE

The drawback of simple password authentication is that the password is sent in plain text. If an unauthorized user is listening, this can compromise the security of the directory because that person can impersonate an authorized user.

The ***nsslapd-require-secure-binds*** configuration attribute requires simple password authentication to occur over a secure connection, using SSL/TLS or Start TLS. This effectively encrypts the plaintext password so it cannot be sniffed by a hacker.

When a secure connection is established between Directory Server and a client application using SSL or the Start TLS operation, the client performs a simple bind with an extra level of protection by not transmitting the password in plaintext. The ***nsslapd-require-secure-binds*** configuration attribute requires simple password authentication over a secure connection, meaning SSL/TLS or Start TLS. This setting allows alternative secure connections, like SASL authentication or certificate-based authentication, as well.

For more information about secure connections, see Section 9.9, “Securing Server Connections”.

9.4.3. Certificate-Based Authentication

An alternative form of directory authentication involves using digital certificates to bind to the directory. The directory prompts users for a password when they first access it. However, rather than matching a password stored in the directory, the password opens the user's certificate database.

If the user supplies the correct password, the directory client application obtains authentication information from the certificate database. The client application and the directory then use this information to identify the user by mapping the user's certificate to a directory DN. The directory allows or denies access based on the directory DN identified during this authentication process.

For more information about certificates and SSL, see the *Administrator's Guide*.

9.4.4. Proxy Authentication

Proxy authentication is a special form of authentication because the user requesting access to the directory does not bind with its own DN but with a *proxy DN*.

The proxy DN is an entity that has appropriate rights to perform the operation requested by the user. When proxy rights are granted to a person or an application, they are granted the right to specify any DN as a proxy DN, with the exception of the Directory Manager DN.

One of the main advantages of proxy right is that an LDAP application can be enabled to use a single thread with a single bind to service multiple users making requests against the Directory Server. Instead of having to bind and authenticate for each user, the client application binds to the Directory Server using a proxy DN.

The proxy DN is specified in the LDAP operation submitted by the client application. For example:

```
ldapmodify -D "cn=directory manager" -W -p 389 -x -D "cn=directory manager" -W -p 389 -h server.example.com -x -Y "cn=joe,dc=example,dc=com" -f mods.ldif
```

This **ldapmodify** command gives the manager entry (**cn=Directory Manager**) the permissions of a user named Joe (**cn=joe**) to apply the modifications in the **mods.ldif** file. The manager does not need to provide Joe's password to make this change.

NOTE

The proxy mechanism is very powerful and must be used sparingly. Proxy rights are granted within the scope of the ACL, and there is no way to restrict who can be impersonated by an entry that has the proxy right. That is, when a user is granted proxy rights, that user has the ability to proxy for any user under the target; there is no way to restrict the proxy rights to only certain users.

For example, if an entity has proxy rights to the **dc=example,dc=com** tree, that entity can do anything. Therefore, ensure that the proxy ACI is set at the lowest possible level of the DIT.

For more information on this topic, check out the "Proxied Authorization ACI Example" section in the "Managing Access Control" chapter of the *Administrator's Guide*.

9.4.5. Pass-through Authentication

Pass-through authentication is when any authentication request is forwarded from one server to another service.

For example, whenever all of the configuration information for an instance is stored in another directory instance, the Directory Server uses pass-through authentication for the User Directory Server to connect to the Configuration Directory Server. Directory Server-to-Directory Server pass-through authentication is handled with the PTA Plug-in.

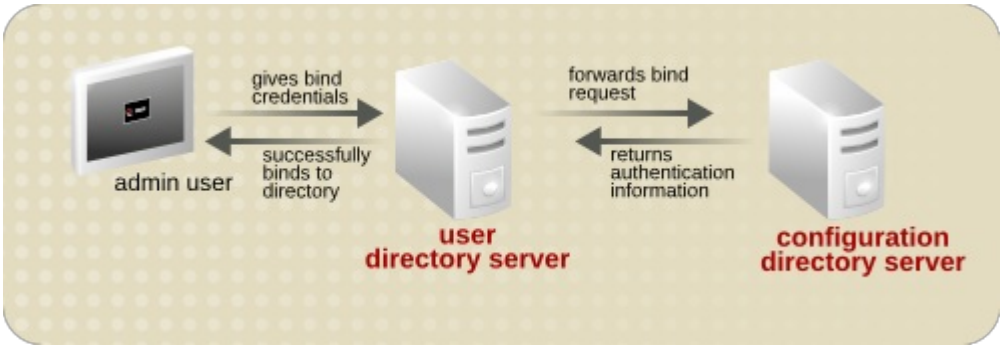


Figure 9.1. Simple Pass-through Authentication Process

Many systems already have authentication mechanisms in place for Unix and Linux users. One of the most common authentication frameworks is *Pluggable Authentication Modules* (PAM). Since many networks already existing authentication services available, administrators may want to continue using those services. A PAM module can be configured to tell Directory Server to use an existing authentication store for LDAP clients.

PAM pass-through authentication in Red Hat Directory Server uses the PAM Pass-through Authentication Plug-in, which enables the Directory Server to talk to the PAM service to authenticate LDAP clients.

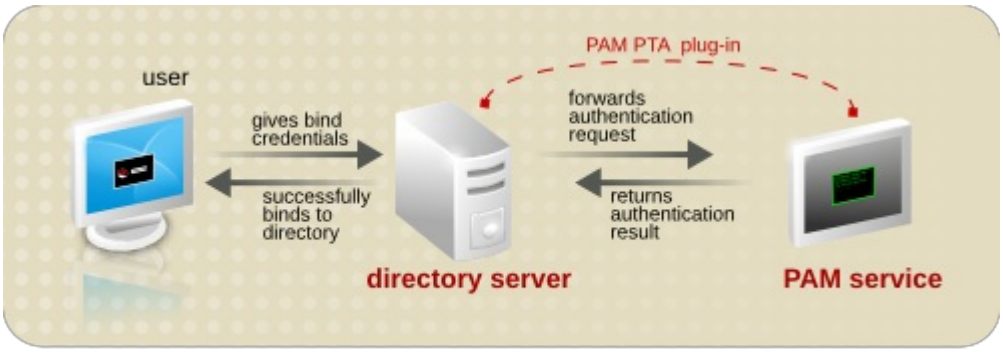


Figure 9.2. PAM Pass-through Authentication Process

With PAM pass-through authentication, when a user attempts to bind to the Directory Server, the credentials are forwarded to the PAM service. If the credentials match the information in the PAM service, then the user can successfully bind to the Directory Server, with all of the Directory Server access control restrictions and account settings in place.

NOTE

The Directory Server can be configured to use PAM, but it cannot be used to *set up* PAM to use the Directory Server for authentication. For PAM to use a Directory Server instance for authentication, the **pam_ldap** module must be properly configured. For general configuration information about **pam_ldap**, look at the manpage (such as http://linux.die.net/man/5/pam_ldap).

The PAM service can be configured using system tools like the *System Security Services Daemon* (SSSD). SSSD can use a variety of different identity providers, including Active Directory, Red Hat Directory Server or other directories like OpenLDAP, or local system settings. To use SSSD, simply point the PAM Pass-through Authentication Plug-in to the PAM file used by SSSD, **/etc/pam.d/system-auth** by default.

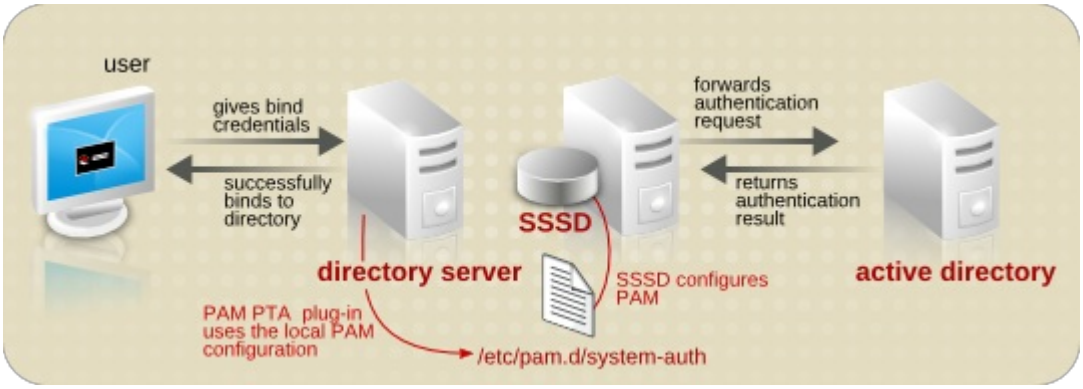


Figure 9.3. PAM Pass-through Authentication with SSSD

9.4.6. Password-less Authentication

An authentication attempt evaluates, first, whether the user account has the ability to authenticate. The account must be active, it must not be locked, and it must have a valid password according to any applicable password policy (meaning it cannot be expired or need to be reset).

There can be times when that *evaluation* of whether a user should be permitted to authenticate needs to be performed, but the user should not (or cannot) be bound to the Directory Server for real. For example, a system may be using PAM to manage system accounts, and PAM is configured to use the LDAP directory as its identity store. However, the system is using password-less credentials, such as SSH keys or RSA tokens, and those credentials cannot be passed to authenticate to the Directory Server.

Red Hat Directory Server supports the *Account Usability Extension Control* for **ldapsearch**es. This control returns information about the account status and any password policies in effect (like requiring a reset, a password expiration warning, or the number of grace logins left after password expiration) – all the information that would be returned in a bind attempt but without authenticating and binding to the Directory Server as that user. That allows the client to determine if the user should be allowed to authenticate based on the Directory Server settings and information, but the actual authentication process is performed outside of Directory Server.

This control can be used with system-level services like PAM to allow password-less logins which still use Directory Server to store identities and even control account status.

NOTE

The Account Usability Extension Control can only be used by the Directory Manager, by default. To allow other users to use the control, set the appropriate ACI on the supported control entry, **oid=1.3.6.1.4.1.42.2.27.9.5.8,cn=features,cn=config**.

9.5. DESIGNING AN ACCOUNT LOCKOUT POLICY

An account lockout policy can protect both directory data and user passwords by preventing unauthorized or compromised access to the directory. After an account has been locked, or *deactivated*, that user cannot bind to the directory, and any authentication operation fails.

Account deactivation is implemented through the operational attribute ***nsAccountLock***. When an entry contains the ***nsAccountLock*** attribute with a value of **true**, the server rejects a bind attempt by that account.

An account lockout policy can be defined based on specific, automatic criteria:

- An account lockout policy can be associated with the password policy (Section 9.6, “Designing a Password Policy”). When a user fails to log in with the proper credentials after a specified number of times, the account is locked until an administrator manually unlocks it.

  This protects against crackers who try to break into the directory by repeatedly trying to guess a user's password.

- An account can be locked after a certain amount of time has lapsed. This can be used to control access for temporary users – such as interns, students, or seasonal workers – who have time-limited access based on the time the account was created. Alternatively, an account policy can be created that inactivates user accounts if the account has been inactive for a certain amount of time since the last login time.

  A time-based account lockout policy is defined through the Account Policy Plug-in, which sets global settings for the directory. Multiple account policy subentries can be created for different expiration times and types and then applied to entries through classes of service.

Additionally, a single user account or a set of accounts (through roles) can be deactivated manually.

NOTE

Deactivating a role deactivates all of the members of that role and not the role entry itself. For more information about roles, see Section 4.3.2, “About Roles”.

9.6. DESIGNING A PASSWORD POLICY

A password policy is a set of rules that govern how passwords are used in a given system. The Directory Server's password policy specifies the criteria that a password must satisfy to be considered valid, like the age, length, and whether users can reuse passwords.

The following sections provide more information on designing a sound password policy:

- Section 9.6.1, “How Password Policy Works”
- Section 9.6.2, “Password Policy Attributes”
- Section 9.6.3, “Designing a Password Policy in a Replicated Environment”

9.6.1. How Password Policy Works

Directory Server supports fine-grained password policy, which means password policies can be defined at the subtree and user level. This allows the flexibility of defining a password policy at any point in the directory tree:

- The entire directory.

  Such a policy is known as the *global* password policy. When configured and enabled, the policy is applied to all users within the directory except for the Directory Manager entry and those user entries that have local password policies enabled.

  This can define a common, single password policy for all directory users.

- A particular subtree of the directory.

  Such a policy is known as the *subtree level* or *local* password policy. When configured and enabled, the policy is applied to all users under the specified subtree.

  This is good in a hosting environment to support different password policies for each hosted company rather than enforcing a single policy for all the hosted companies.

- A particular user of the directory.

  Such a policy is known as the *user level* or *local* password policy. When configured and enabled, the policy is applied to the specified user only.

  This can define different password policies for different directory users. For example, specify that some users change their passwords daily, some users change it monthly, and all other users change it every six months.

By default, Directory Server includes entries and attributes that are relevant to the global password policy, meaning the same policy is applied to all users. To set up a password policy for a subtree or user, add additional entries at the subtree or user level and enable the ***nsslapd-pwpolicy-local*** attribute of the **cn=config** entry. This attribute acts as a switch, turning fine-grained password policy on and off.

The password policy changes can be made in the Directory Server Console or by using the **ns-newpwpolicy.pl** script. The *Configuration, Command, and File Reference* lists the command-line syntax for the script, and the *Administrator's Guide* includes procedures for setting password policies.

After password policy entries are added to the directory, they determine the type (global or local) of the password policy the Directory Server should enforce.

When a user attempts to bind to the directory, Directory Server determines whether a local policy has been defined and enabled for the user's entry.

- To determine whether the fine-grained password policy is enabled, the server checks the value (**on** or **off**) assigned to the ***nsslapd-pwpolicy-local*** attribute of the **cn=config** entry. If the value is **off**, the server ignores the policies defined at the subtree and user levels and enforces the global password policy.

- To determine whether a local policy is defined for a subtree or user, the server checks for the ***pwdPolicysubentry*** attribute in the corresponding user entry. If the attribute is present, the server enforces the local password policy configured for the user. If the attribute is absent, the server logs an error message and enforces the global password policy.

The server then compares the user-supplied password with the value specified in the user's directory entry to make sure they match. The server also uses the rules defined by the password policy to ensure that the password is valid before allowing the user to bind to the directory.

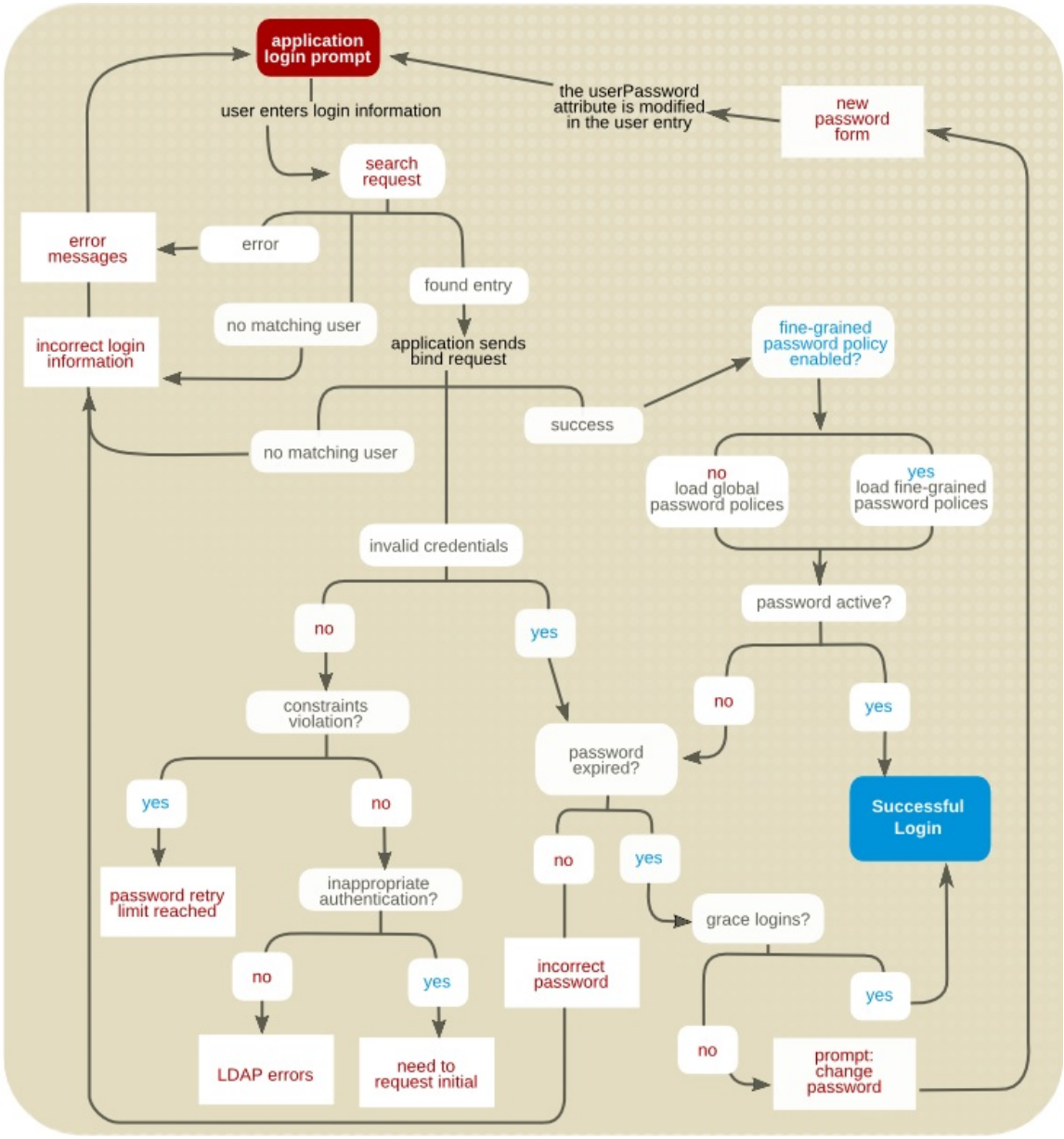


Figure 9.4. Password Policy Checking Process

In addition to bind requests, password policy checking also occurs during add and modify operations if the ***userPassword*** attribute (explained in the following section) is present in the request.

Modifying the value of ***userPassword*** checks two password policy settings:

- The password minimum age policy is activated. If the minimum age requirement has not been satisfied, the server returns a constraintViolation error. The password update operation fails.
- The password history policy is activated. If the new value of ***userPassword*** is in the password history, or if it is the same as the current password, the server returns a constraintViolation error. The password update operation fails.

Both adding and modifying the value of ***userPassword*** checks password policies set for the password syntax:

- The password minimum length policy is activated. If the new value of ***userPassword*** is less than the required minimum length, the server returns a constraintViolation error. The password update operation fails.
- The password syntax checking policy is activated. If the new value of ***userPassword*** is the same as another attribute of the entry, the server returns a constraintViolation error. The password update operation fails.

9.6.2. Password Policy Attributes

The following sections describe the attributes to create a password policy for the server:

- Section 9.6.2.1, “Maximum Number of Failures”

- Section 9.6.2.2, "Password Change After Reset"
- Section 9.6.2.3, "User-Defined Passwords"
- Section 9.6.2.4, "Password Expiration"
- Section 9.6.2.5, "Expiration Warning"
- Section 9.6.2.6, "Grace Login Limit"
- Section 9.6.2.7, "Password Syntax Checking"
- Section 9.6.2.8, "Password Length"
- Section 9.6.2.9, "Password Minimum Age"
- Section 9.6.2.10, "Password History"
- Section 9.6.2.11, "Password Storage Schemes"
- Section 9.6.2.12, "Password Last Change Time"

See the *Red Hat Directory Server Administrator's Guide* for instructions on how to set these attributes.

9.6.2.1. Maximum Number of Failures

This is a setting in the password policy which enables password-based account lockouts. If a user attempts to log in a certain number of times and fails, then that account is locked until an administrator unlocks it or, optionally, a certain amount of time passes. This is set in the ***passwordMaxFailure*** parameter.

There are two different ways to count login attempts when evaluating when the maximum number of failed attempts is reached. It can be a hard limit which locks the account when the number is hit (*n*) or which locks the account only when the count is exceeded (*n+1*). For example, if the failure limit is three attempts, then the account could be locked at the third failed attempt (*n*) or at the fourth failed attempt (*n+1*). The *n+1* behavior is the historical behavior for LDAP servers, so it is considered legacy behavior. Newer LDAP clients expect the stricter hard limit. By default, the Directory Server uses the strict limit (*n*), but the legacy behavior can be enabled in the ***passwordLegacyPolicy*** parameter.

9.6.2.2. Password Change After Reset

The Directory Server password policy can specify whether users must change their passwords after the first login or after the password has been reset by the administrator.

The default passwords set by the administrator typically follow a company convention, such as the user's initials, user ID, or the company name. If this convention is discovered, it is usually the first value that a cracker uses in an attempt to break into the system. It is therefore recommended that users be required to change their password after it has been reset by an administrator. If this option is configured for the password policy, users are required to change their password even if user-defined passwords are disabled.

If users are not required or allowed change their own passwords, administrator-assigned passwords should not follow any obvious convention and should be difficult to discover.

The default configuration does not require that users change their password after it has been reset.

See Section 9.6.2.3, "User-Defined Passwords" for more information.

9.6.2.3. User-Defined Passwords

The password policy can be set either to allow or not to allow users to change their own passwords. A good password is the key to a strong password policy. Good passwords do not use trivial words; any word that can be found in a dictionary, names of pets or children, birthdays, user IDs, or any other information about the user that can be easily discovered (or stored in the directory itself), is a poor choice for a password.

A good password should contain a combination of letters, numbers, and special characters. For the sake of convenience, however, users often use passwords that are easy to remember. Consequently, some enterprises choose to set passwords for users that meet the criteria of a strong password, and do not allow users to change their passwords.

There are two disadvantages to having administrators set passwords for users:

- It requires a substantial amount of an administrator's time.

- Because administrator-specified passwords are typically more difficult to remember, users are more likely to write their password down, increasing the risk of discovery.

By default, user-defined passwords are allowed.

9.6.2.4. Password Expiration

The password policy can allow users can use the same passwords indefinitely or specify that passwords expire after a given time. In general, the longer a password is in use, the more likely it is to be discovered. If passwords expire too often, however, users may have trouble remembering them and resort to writing their passwords down. A common policy is to have passwords expire every 30 to 90 days.

The server remembers the password expiration specification even if password expiration is disabled. If the password expiration is re-enabled, passwords are valid only for the duration set before it was last disabled.

For example, if the password policy is set for passwords to expire every 90 days, and then password expiration is disabled and re-enabled, the default password expiration duration is 90 days.

By default, user passwords never expire.

9.6.2.5. Expiration Warning

If a password expiration period is set, it is a good idea to send users a warning before their passwords expire.

The Directory Server displays the warning when the user binds to the server. If password expiration is enabled, by default, a warning is sent (using an LDAP message) to the user one day before the user's password expires, provided the user's client application supports this feature.

The valid range for a password expiration warning to be sent is from one to 24,855 days.

NOTE

The password *never* expires until the expiration warning has been sent.

9.6.2.6. Grace Login Limit

A *grace period* for expired passwords means that users can still log in to the system, even if their password has expired. To allow some users to log in using an expired password, specify the number of grace login attempts that are allowed to a user after the password has expired.

By default, grace logins are not permitted.

9.6.2.7. Password Syntax Checking

Password syntax checking enforces rules for password strings, so that any password has to meet or exceed certain criteria. All password syntax checking can be applied globally, per subtree, or per user. Password syntax checking is set in the ***passwordCheckSyntax*** attribute.

The default password syntax requires a minimum password length of eight characters and that no trivial words are used in the password. A trivial word is any value stored in the ***uid***, ***cn***, ***sn***, ***givenName***, ***ou***, or ***mail***attributes of the user's entry.

Additionally, other forms of password syntax enforcement are possible, providing different optional categories for the password syntax:

- Minimum required number of characters in the password (***passwordMinLength***)
- Minimum number of digit characters, meaning numbers between zero and nine (***passwordMinDigits***)
- Minimum number of ASCII alphabetic characters, both upper- and lower-case (***passwordMinAlphas***)
- Minimum number of uppercase ASCII alphabetic characters (***passwordMinUppers***)
- Minimum number of lowercase ASCII alphabetic characters (***passwordMinLowers***)
- Minimum number of special ASCII characters, such as **!@#$** (***passwordMinSpecials***)
- Minimum number of 8-bit characters (***passwordMin8bit***)
- Maximum number of times that the same character can be immediately repeated, such as **aaabbb** (***passwordMaxRepeats***)

- Minimum number of character categories required per password; a category can be upper- or lower-case letters, special characters, digits, or 8-bit characters (***passwordMinCategories***)

The more categories of syntax required, the stronger the password.

By default, password syntax checking is disabled.

9.6.2.8. Password Length

The password policy can require a minimum length for user passwords. In general, shorter passwords are easier to crack. A good length for passwords is eight characters. This is long enough to be difficult to crack but short enough that users can remember the password without writing it down. The valid range of values for this attribute is from two to 512 characters.

By default, no minimum password length is set.

9.6.2.9. Password Minimum Age

The password policy can prevent users from changing their passwords for a specified time. When used in conjunction with the ***passwordHistory*** attribute, users are discouraged from reusing old passwords.

For example, if the password minimum age (***passwordMinAge***) attribute is two days, users cannot repeatedly change their passwords during a single session. This prevents them from cycling through the password history so that they can reuse an old password.

The valid range of values for this attribute is from zero to 24,855 days. A value of zero (0) indicates that the user can change the password immediately.

9.6.2.10. Password History

The Directory Server can store from two to 24 passwords in the *password history*; if a password is in the history, a user cannot reset his password to that old password. This prevents users from reusing a couple of passwords that are easy to remember. Alternatively, the password history can be disabled, thus allowing users to reuse passwords.

The passwords remain in history even if the password history is off so that if the password history is turned back on, users cannot reuse the passwords that were in the history before the password history was disabled.

The server does not maintain a password history by default.

9.6.2.11. Password Storage Schemes

The password storage scheme specifies the type of encryption used to store Directory Server passwords within the directory. The Directory Server supports several different password storage schemes:

- *Salted Secure Hash Algorithm* (SSHA, SSHA-256, SSHA-384, and SSHA-512). This is the most secure password storage scheme and is the default. The recommended SSHA scheme is SSHA-256 or stronger.
- *CLEAR*, meaning no encryption. This is the only option which can be used with SASL Digest-MD5, so using SASL requires the CLEAR password storage scheme.

  Although passwords stored in the directory can be protected through the use of access control information (ACI) instructions, it is still not a good idea to store plain text passwords in the directory.
- *Secure Hash Algorithm* (SHA, SHA-256, SHA-384, and SHA-512). This is less secure than SSHA.
- *UNIX CRYPT algorithm*. This algorithm provides compatibility with UNIX passwords.
- *MD5*. This storage scheme is less secure than SSHA, but it is included for legacy applications which require MD5.
- *Salted MD5*. This storage scheme is more secure than plain MD5 hash, but still less secure than SSHA. This storage scheme is not included for use with new passwords but to help with migrating user accounts from directories which support salted MD5.

9.6.2.12. Password Last Change Time

The ***passwordTrackUpdateTime*** attribute tells the server to record a timestamp for the last time that the password was updated for an entry. The password change time itself is stored as an operational attribute on the user entry, ***pwdUpdateTime*** (which is separate from the ***modifyTimestamp*** or ***lastModified*** operational attributes).

By default, the password change time is *not* recorded.

9.6.3. Designing a Password Policy in a Replicated Environment

Password and account lockout policies are enforced in a replicated environment as follows:

- Password policies are enforced on the data master.
- Account lockout is enforced on all servers in the replication setup.

The password policy information in the directory, such as password age; the account lockout counter; and the expiration warning counter are all replicated. The configuration information, however, is stored locally and is not replicated. This information includes the password syntax and the history of password modifications.

When configuring a password policy in a replicated environment, consider the following points:

- All replicas issue warnings of an impending password expiration. This information is kept locally on each server, so if a user binds to several replicas in turn, the user receives the same warning several times. In addition, if the user changes the password, it may take time for this information to filter through to the replicas. If a user changes a password and then immediately rebinds, the bind may fail until the replica registers the changes.
- The same bind behavior should occur on all servers, including suppliers and replicas. Always create the same password policy configuration information on each server.
- Account lockout counters may not work as expected in a multi-master environment.

9.7. DESIGNING ACCESS CONTROL

After deciding on the authentication schemes to use to establish the identity of directory clients, decide how to use those schemes to protect the information contained in the directory. Access control can specify that certain clients have access to particular information, while other clients do not.

Access control is defined using one or more *access control lists* (ACLs). The directory's ACLs consist of a series of one or more *access control information* (ACI) statements that either allow or deny permissions (such as read, write, search, and compare) to specified entries and their attributes.

Using the ACL, permissions can be set at any level of the directory tree:

- The entire directory.
- A particular subtree of the directory.
- Specific entries in the directory.
- A specific set of entry attributes.
- Any entry that matches a given LDAP search filter.

In addition, permissions can be set for a specific user, for all users belonging to a specific group, or for all users of the directory. Lastly, access can be defined for a network location such as an IP address (IPv4 or IPv6) or a DNS name.

9.7.1. About the ACI Format

When designing the security policy, it is helpful to understand how ACIs are represented in the directory. It is also helpful to understand what permissions can be set in the directory. This section gives a brief overview of the ACI mechanism. For a complete description of the ACI format, see the *Red Hat Directory Server Administrator's Guide* .

Directory ACIs use the following general form: *target permission bind_rule*

The ACI variables are defined below:

- *target*. Specifies the entry (usually a subtree) that the ACI targets, the attribute it targets, or both. The target identifies the directory element that the ACI applies to. An ACI can target only one entry, but it can target multiple attributes. In addition, the target can contain an LDAP search filter. Permissions can be set for widely scattered entries that contain common attribute values.
- *permission*. Identifies the actual permission being set by this ACI. The permission variable states that the ACI is allowing or denying a specific type of directory access, such as read or search, to the specified target.
- *bind rule*. Identifies the bind DN or network location to which the permission applies. The bind rule may also specify an LDAP filter, and if that filter is evaluated to be true for the binding client application, then the ACI applies to the client application.

ACIs can therefore be expressed as follows: "For the directory object target, allow or deny permission if bind_rule is true."

permission and *bind_rule* are set as a pair, and there can be multiple *permission-bind_rule* pairs for every target. Multiple access controls can be effectively set for any given target. For example:

```
target (permission bind_rule)(permission bind_rule)...
```

A permission can be set to allow anyone binding as Babs Jensen to write to Babs Jensen's telephone number. The bind rule in this permission is the part that states *"if you bind as Babs Jensen."* The target is Babs Jensen's phone number, and the permission is *write* access.

9.7.1.1. Targets

Decide which entry is targeted by every ACI created in the directory. Targeting a directory branch point entry includes that branch point and all of its child entries in the scope of the permission. If a target entry is not explicitly defined for the ACI, then the ACI is targeted to the directory entry that contains the ACI statement. Set the ***targetattr*** parameter to target one or more attributes. If the ***targetattr*** parameter is not set, no attributes are targeted. For further details, see the corresponding section in the *Red Hat Directory Server Administration Guide*.

For every ACI, only one entry or only those entries that match a single LDAP search filter can be targeted.

In addition to targeting entries, it is possible to target attributes on the entry; this applies the permission to only a subset of attribute values. Target sets of attributes by explicitly naming those attributes that are targeted or by explicitly naming the attributes that are not targeted by the ACI. Excluding attributes in the target sets a permission for all but a few attributes allowed by an object class structure.

For further details, see the corresponding section in the *Red Hat Directory Server Administration Guide*.

9.7.1.2. Permissions

Permissions can either allow or deny access. In general, avoid denying permissions (for the reasons explained in Section 9.7.2.2, “Allowing or Denying Access”). Permissions can be any operation performed on the directory service:

Permission	Description
Read	Indicates whether directory data may be read.
Write	Indicates whether directory data may be changed or created. This permission also allows directory data to be deleted but not the entry itself. To delete an entire entry, the user must have delete permissions.
Search	Indicates whether the directory data can be searched. This differs from the read permission in that read allows directory data to be viewed if it is returned as part of a search operation. For example, if searching for common names is allowed as well as read permission for a person's room number, then the room number can be returned as part of the common name search, but the room number itself cannot be used as the subject of a search. Use this combination to prevent people from searching the directory to see who sits in a particular room.
Compare	Indicates whether the data may be used in comparison operations. The compare permission implies the ability to search, but actual directory information is not returned as a result of the search. Instead, a simple Boolean value is returned which indicates whether the compared values match. This is used to match ***userPassword*** attribute values during directory authentication.
Self-write	Used only for group management. This permission enables a user to add to or delete themselves from a group.

Permission	Description
Add	Indicates whether child entries can be created. This permission enables a user to create child entries beneath the targeted entry.
Delete	Indicates whether an entry can be deleted. This permission enables a user to delete the targeted entry.
Proxy	Indicates that the user can use any other DN, except Directory Manager, to access the directory with the rights of this DN.

9.7.1.3. Bind Rules

The bind rule usually indicates the bind DN subject to the permission. It can also specify bind attributes such as time of day or IP address.

Bind rules easily express that the ACI applies only to a user's own entry. This allows users to update their own entries without running the risk of a user updating another user's entry.

Bind rules indicate that the ACI is applicable in specific situations:

- Only if the bind operation is arriving from a specific IP address (IPv4 or IPv6) or DNS host name. This is often used to force all directory updates to occur from a given machine or network domain.
- If the person binds anonymously. Setting a permission for anonymous bind also means that the permission applies to anyone who binds to the directory as well.
- For anyone who successfully binds to the directory. This allows general access while preventing anonymous access.
- Only if the client has bound as the immediate parent of the entry.
- Only if the entry as which the person has bound meets a specific LDAP search criteria.

The Directory Server provides several keywords to more easily express these kinds of access:

- *Parent*. If the bind DN is the immediate parent entry, then the bind rule is true. This means that specific permissions can be granted that allow a directory branch point to manage its immediate child entries.
- *Self*. If the bind DN is the same as the entry requesting access, then the bind rule is true. Specific permission can be granted to allow individuals to update their own entries.
- *All*. The bind rule is true for anyone who has successfully bound to the directory.
- *Anyone*. The bind rule is true for everyone. This keyword is used to allow or deny anonymous access.

9.7.2. Setting Permissions

By default, all users are denied access rights of any kind, with the exception of the Directory Manager. Consequently, some ACIs must be set for the directory for users to be able to access the directory.

For information about how to set ACIs in the directory, see the *Red Hat Directory Server Administrator's Guide* .

9.7.2.1. The Precedence Rule

When a user attempts any kind of access to a directory entry, Directory Server examines the access control set in the directory. To determine access, Directory Server applies the *precedence rule*. This rule states that when two conflicting permissions exist, the permission that denies access always takes precedence over the permission that grants access.

For example, if write permission is denied at the directory's root level, and that permission is applied to everyone accessing the directory, then no user can write to the directory regardless of any other permissions that may allow write access. To allow a specific user write permissions to the directory, the scope of the original deny-for-write has to be set so that it does not include that user. Then, there must be additional allow-for-write permission for the user in question.

9.7.2.2. Allowing or Denying Access

Access to the directory tree can be explicitly allowed or denied, but be careful of explicitly denying access to the directory. Because of the precedence rule, if the directory finds rules explicitly forbidding access, the directory forbids access regardless of any conflicting permissions that may grant access.

Limit the scope of allow access rules to include only the smallest possible subset of users or client applications. For example, permissions can be set that allow users to write to any attribute on their directory entry, but then deny all users except members of the Directory Administrators group the privilege of writing to the ***uid*** attribute. Alternatively, write two access rules that allow write access in the following ways:

- Create one rule that allows write privileges to every attribute except the ***uid*** attribute. This rule should apply to everyone.
- Create one rule that allows write privileges to the ***uid*** attribute. This rule should apply only to members of the Directory Administrators group.

Providing only allow privileges avoids the need to set an explicit deny privilege.

9.7.2.3. When to Deny Access

It is rarely necessary to set an explicit deny privilege, but there are a few circumstances where it is useful:

- There is a large directory tree with a complex ACL spread across it.

  For security reasons, it may be necessary to suddenly deny access to a particular user, group, or physical location. Rather than spending the time to carefully examine the existing ACL to understand how to appropriately restrict the allow permissions, temporarily set the explicit deny privilege until there is time to do the analysis. If the ACL has become this complex, then, in the long run, the deny ACI only adds to the administrative overhead. As soon as possible, rework the ACL to avoid the explicit deny privilege and then simplify the overall access control scheme.

- Access control should be based on a day of the week or an hour of the day.

  For example, all writing activities can be denied from Sunday at 11:00 p.m. (2300) to Monday at 1:00 a.m. (0100). From an administrative point of view, it may be easier to manage an ACI that explicitly restricts time-based access of this kind than to search through the directory for all the allow-for-write ACIs and restrict their scopes in this time frame.

- Privileges should be restricted when delegating directory administration authority to multiple people.

  To allow a person or group of people to manage some part of the directory tree, without allowing them to modify some aspect of the tree, use an explicit deny privilege.

  For example, to make sure that Mail Administrators do not allow write access to the common name attribute, then set an ACI that explicitly denies write access to the common name attribute.

9.7.2.4. Where to Place Access Control Rules

Access control rules can be placed on any entry in the directory. Often, administrators place access control rules on entries with the object classes **domainComponent**, **country**, **organization**, **organizationalUnit**, **inetOrgPerson**, or **group**.

Organize rules into groups as much as possible in order to simplify ACL administration. Rules generally apply to their target entry and to all of that entry's children. Consequently, it is best to place access control rules on root points in the directory or on directory branch points, rather than scatter them across individual leaf (such as person) entries.

9.7.2.5. Using Filtered Access Control Rules

One of the more powerful features of the Directory Server ACI model is the ability to use LDAP search filters to set access control. Use LDAP search filters to set access to any directory entry that matches a defined set of criteria.

For example, allow read access for any entry that contains an ***organizationalUnit*** attribute that is set to Marketing.

Filtered access control rules allow predefined levels of access. Suppose the directory contains home address and telephone number information. Some people want to publish this information, while others want to be unlisted. There are several ways to address that:

- Create an attribute on every user's directory entry called ***publishHomeContactInfo***.
- Set an access control rule that grants read access to the ***homePhone*** and ***homePostalAddress*** attributes only for entries whose ***publishHomeContactInfo*** attribute is set to **true** (meaning enabled). Use an LDAP search filter to express the target for this rule.

- Allow the directory users to change the value of their own ***publishHomeContactInfo*** attribute to either **true** or **false**. In this way, the directory user can decide whether this information is publicly available.

For more information about using LDAP search filters and on using LDAP search filters with ACIs, see the *Red Hat Directory Server Administrator's Guide*.

9.7.3. ACIs and the Directory Manager

f

9.7.4. Viewing ACIs: Get Effective Rights

It can be necessary to view access controls set on an entry to grant fine-grained access control or for efficient entry management. *Get effective rights* is an extended **ldapsearch** which returns the access control permissions set on each attribute within an entry, and allows an LDAP client to determine what operations the server's access control configuration allows a user to perform.

The access control information is divided into two groups of access: rights for an entry and rights for an attribute. "Rights for an entry" means the rights, such as modify or delete, that are limited to that specific entry. "Rights for an attribute" means the access right to every instance of that attribute throughout the directory.

This kind of detailed access control may be necessary in the following types of situations:

- An administrator can use the *get effective rights* command for minute access control, such as allowing certain groups or users access to entries and restricting others. For example, members of the QA Managers group may have the right to search and read attributes such as ***title*** and ***salary***, but only HR Group members have the rights to modify or delete them.

- A user can use the get effective rights option to determine what attributes they can view or modify on their personal entry. For example, a user should have access to attributes such as ***homePostalAddress*** and ***cn***, but may only have read access to ***title*** and ***salary***.

An **ldapsearch** executed using the **-E** switch returns the access controls on a particular entry as part of the normal search results. The following search the rights that user Ted Morris has to his personal entry:

```
ldapsearch -x -p 389 -h server.example.com -D "uid=tmorris,ou=people,dc=example,dc=com" -w password -b
"uid=tmorris,ou=people,dc=example,dc=com" -E
!1.3.6.1.4.1.42.2.27.9.5.2:dn:uid=tmorris,ou=people,dc=example,dc=com "(objectClass=*)"

version: 1
dn: uid=tmorris,ou=People,dc=example,dc=com
givenName: Ted
sn: Morris
ou: Accounting
ou: People
l: Santa Clara
manager: uid=dmiller,ou=People,dc=example,dc=com
roomNumber: 4117
mail: tmorris@example.com
facsimileTelephoneNumber: +1 408 555 5409
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
uid: tmorris
cn: Ted Morris
userPassword: {SSHA}bz0uCmHZM5b357zwrCUCJs1IOHtMD6yqPyhxBA==
entryLevelRights: vadn
attributeLevelRights: givenName:rsc, sn:rsc, ou:rsc, l:rscow, manager:rsc, roomNumber:rscwo, mail:rscwo,
facsimileTelephoneNumber:rscwo, objectClass:rsc, uid:rsc, cn:rsc, userPassword:wo
```

In this example, Ted Morris has the right to add, view, delete, or rename the DN on his own entry, as shown by the results in ***entryLevelRights***. He can read, search, compare, self-modify, or self-delete the location (***l***) attribute but only self-write and self-delete rights to his password, as shown in the ***attributeLevelRights*** result.

By default, effective rights information is not returned for attributes in an entry that do not have a value or which do not exist in the entry. For example, if the ***userPassword*** value is removed, then a future effective rights search on the above entry would not return any effective rights for ***userPassword***, even though self-write and self-delete rights could be allowed. Similarly, if the ***street*** attribute were added with read, compare, and search rights, then ***street: rsc*** would appear in the ***attributeLevelRights*** results.

It is possible to return rights for attributes which are not normally included in the search results, like non-existent attributes or operational attributes. Using an asterisk (*****) returns the rights for all possible attributes for an entry, including non-existent attributes.

```
ldapsearch -x -E !1.3.6.1.4.1.42.2.27.9.5.2:dn:uid=scarter,ou=people,dc=example,dc=com "(objectclass=*)" "*"
```

Using the plus sign (**+**) returns operational attributes for the entry, which are not normally returned in an **ldapsearch** asterisk (*). For example:

```
ldapsearch -x -E !1.3.6.1.4.1.42.2.27.9.5.2:dn:uid=scarter,ou=people,dc=example,dc=com "(objectclass=*)" "+"
```

The asterisk (*****) and the plus sign (**+**) can be used together to return every attribute for the entry.

Get effective rights for existing attributes are also visible in the Directory Server Console. Open the **Advanced Properties** editor for the user entry, and then select the **Show effective rights** check box. This displays the attribute-level rights (**r**, **s**, **c**, **w**, **o**) next to the attributes listed in the main window and the entry-level rights (**v**, **a**, **d**, **n**) underneath the entry's DN at the bottom of the window.

9.7.5. Using ACIs: Some Hints and Tricks

Keep this tips in mind when implementing the security policy. They can help to lower the administrative burden of managing the directory security model and improve the directory's performance characteristics.

- Minimize the number of ACIs in the directory.

  Although the Directory Server can evaluate over 50,000 ACIs, it is difficult to manage a large number of ACI statements. A large number of ACIs makes it hard for human administrators to immediately determine the directory object available to particular clients.

  Directory Server minimizes the number of ACIs in the directory by using macros. Macros are placeholders that are used to represent a DN, or a portion of a DN, in an ACI. Use the macro to represent a DN in the target portion of the ACI or in the bind rule portion, or both. For more information on macro ACIs, see the "Managing Access Control" chapter in the *Red Hat Directory Server Administrator's Guide* .

- Balance allow and deny permissions.

  Although the default rule is to deny access to any user who has not been specifically granted access, it may be better to reduce the number of ACIs by using one ACI to allow access close to the root of the tree, and a small number of deny ACIs close to the leaf entries. This scenario can avoid the use of multiple allow ACIs close to the leaf entries.

- Identify the smallest set of attributes on any given ACI.

  When allowing or denying access to a subset of attributes on an object, determine whether the smallest list is the set of attributes that are allowed or the set of attributes that are denied. Then express the ACI so that it only requires managing the smallest list.

  For example, the **person** object class contains a large number of attributes. To allow a user to update only one or two of these attributes, write the ACI so that it allows write access for only those few attributes. However, to allow a user to update all but one or two attributes, create the ACI so that it allows write access for everything but a few named attributes.

- Use LDAP search filters cautiously.

  Search filters do not directly name the object for which you are managing access. Consequently their use can produce unexpected results. This is especially true as the directory becomes more complex. Before using search filters in ACIs, run an **ldapsearch** operation using the same filter to make clear what the results of the changes mean to the directory.

- Do not duplicate ACIs in differing parts of the directory tree.

  Guard against overlapping ACIs. For example, if there is an ACI at the directory root point that allows a group write access to the ***commonName*** and ***givenName*** attributes, and another ACI that allows the same group write access for only the ***commonName*** attribute, then consider reworking the ACIs so that only one control grants the write access for the group.

  As the directory grows more complex, the risk of accidentally overlapping ACIs quickly increases. By avoiding ACI overlap, security management becomes easier while potentially reducing the total number of ACIs contained in the directory.

- Name ACIs.

While naming ACIs is optional, giving each ACI a short, meaningful name helps with managing the security model, especially when examining ACIs from the Directory Server Console.

- Group ACIs as closely together as possible within the directory.

  Try to limit ACI placement to the directory root point and to major directory branch points. Grouping ACIs helps to manage the total list of ACIs, as well as helping keep the total number of ACIs in the directory to a minimum.

- Avoid using double negatives, such as *deny write if the bind DN is not equal to cn=Joe* .

  Although this syntax is perfectly acceptable for the server, it is confusing for a human administrator.

9.7.6. Applying ACIs to the Root DN (Directory Manager)

Normally, access control rules do not apply to the Directory Manager user. The Directory Manager is defined in the **dse.ldif** file, not in the regular user database, and so ACI targets do not include that user.

It also makes sense from a maintenance perspective. The Directory Manager requires a high level of access in order to perform maintenance tasks and to response to incidents.

Still, because of the power of the Directory Manager user, a certain level of access control may be advisable to prevent unauthorized access or attacks from being performed as the root user.

The RootDN Access Control Plug-in sets certain access control rules specific to the Directory Manager user:

- Time-based access controls for time ranges, such as 8a.m. to 5p.m. (0800 to 1700).
- Day-of-week access controls, so access is only allowed on explicitly defined days
- IP address rules, where only specified IP addresses, domains, or subnets are explicitly allowed or denied
- Host access rules, where only specified host names, domain names, or subdomains are explicitly allowed or denied

As with other access control rules, deny rules supercede allow rules.

IMPORTANT

Make sure that the Directory Manager always has the approproate level of access allowed. The Directory Manager may need to perform maintenance operations in off-hours (when user load is light) or to respond to failures. In that case, setting stringent time or day-based access control rules could prevent the Directory Manager from being able to adequately manage the directory.

Root DN access control rules are disabled by default. The RootDN Access Control Plug-in must be enabled, and then the appropriate access control rules can be set.

NOTE

There is only *one* access control rule set for the Directory Manager, in the plug-in entry, and it applies to all access to the entire directory.

9.8. ENCRYPTING THE DATABASE

Information is stored in a database in plain text. Consequently, some extremely sensitive information, such as government identification numbers or passwords, may not be sufficiently protected by access control measures. It may be possible to gain access to a server's persistent storage files, either directly through the file system or by accessing discarded disk drives or archive media.

Database encryption allows individual attributes to be encrypted as they are stored in the database. When configured, every instance of a particular attribute, even index data, is encrypted and can only be accessed using a secure channel, such as SSL/TLS.

For information on using database encryption, see the "Configuring Directory Databases" chapter in the *Red Hat Directory Server Administrator's Guide*.

9.9. SECURING SERVER CONNECTIONS

After designing the authentication scheme for identified users and the access control scheme for protecting information in the directory, the next step is to design a way to protect the integrity of the information as it passes between servers and client applications.

For both server to client connections and server to server connections, the Directory Server supports a variety of secure connection types:

- *Secure Sockets Layer (SSL) and Transport Layer Security (TLS)* .

  To provide secure communications over the network, the Directory Server can use LDAP over the Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

  SSL/TLS can be used in conjunction with the RC2 and RC4 encryption algorithms from RSA. The encryption method selected for a particular connection is the result of a negotiation between the client application and Directory Server.

- *Start TLS*.

  Directory Server also supports Start TLS, a method of initiating a Transport Layer Security (TLS) connection over a regular, unencrypted LDAP port.

- *Simple Authentication and Security Layer (SASL)* .

  SASL is a security framework, meaning that it sets up a system that allows different mechanisms to authenticate a user to the server, depending on what mechanism is enabled in both client and server applications. It can also establish an encrypted session between the client and a server. In Directory Server, SASL is used with GSS-API to enable Kerberos logins and can be used for almost all server to server connections, including replication, chaining, and pass-through authentication. (SASL cannot be used with Windows Sync.)

Secure connections are recommended for any operations which handle sensitive information, like replication, and are required for some operations, like Windows password synchronization. Directory Server can support SSL/TLS connections, SASL, and non-secure connections simultaneously.

Both SASL authentication and SSL/TLS connections can be configured at the same time. For example, the Directory Server instance can be configured to require SSL connections to the server and also support SASL authentication for replication connections. This means it is not necessary to choose whether to use SSL/TLS *or* SASL in a network environment; you can use both.

It is also possible to set a minimum level of security for connections to the server. The *security strength factor* measures, in key strength, how strong a secure connection is. An ACI can be set that requires certain operations (like password changes) only occur if the connection is of a certain strength or higher. It is also possible to set a minimum SSF, which can essentially disable standard connections and requires SSL/TLS, Start TLS, or SASL for every connection. The Directory Server supports SSL/TLS and SASL simultaneously, and the server calculates the SSF of all available connection types and selects the strongest.

For more information about using SSL/TLS, Start TLS, and SASL, check out the *Administrator's Guide*.

9.10. USING SELINUX POLICIES

Although Security-Enhanced Linux is a security feature on a Red Hat Enterprise Linux machine, the Directory Server and its Admin Server and SNMP components have special SELinux policies in place to make the server run effectively in a secure system environment.

SELinux is a collection of mandatory access control rules which are enforced across a system to restrict unauthorized access and tampering. SELinux categorizes files, directories, ports, processes, users, and other objects on the server. Each object is placed in an appropriate security context to define how the object is allowed to behave on the server through its role, user, and security level. These roles for objects are grouped in domains, and SELinux rules define how the objects in one domain are allowed to interact with objects in another domain.

Directory Server has three domains:

- dirsrv_t for the Directory Server
- dirsrvadmin_t for the Admin Server
- dirsrv_snmp_t for the SNMP

Directory Server also uses two additional, default domains for LDAP ports (ldap_port_t) and web services (httpd_t).

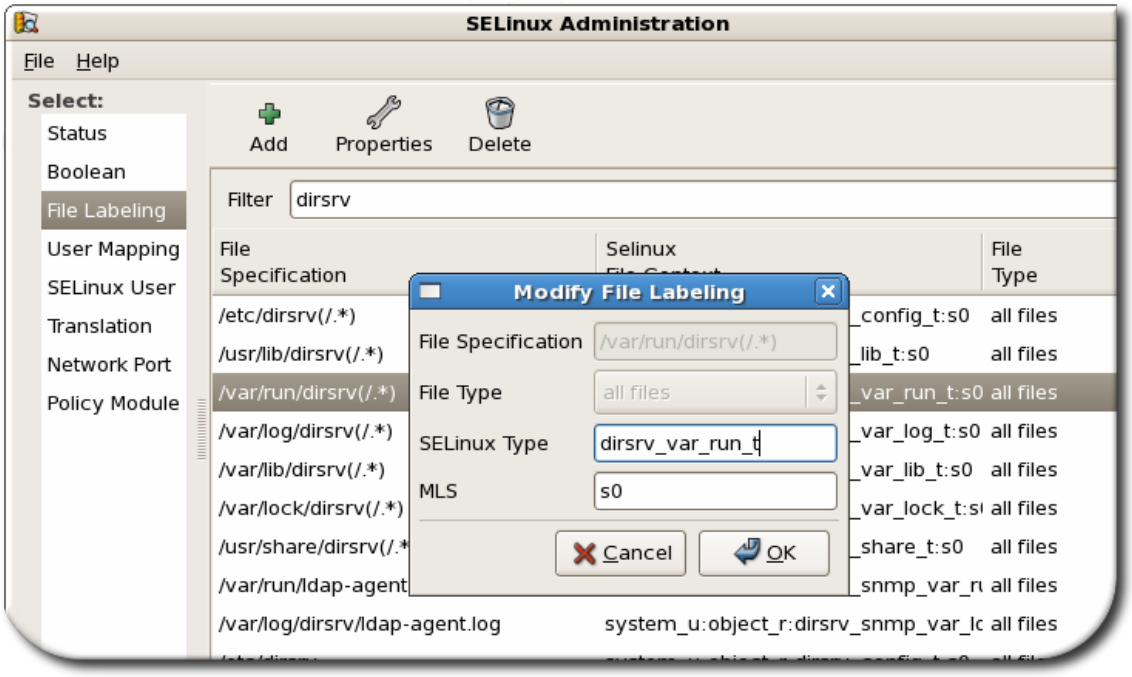


Figure 9.5. Editing Directory Server File Labeling

These domains provide security contexts for all of the processes, files, directories, ports, sockets, and users for the Directory Server.

- Files and directories for each instance are labeled with a specific SELinux context. (Most of the main directories used by Directory Server have subdirectories for all local instances, no matter how many, so a single policy is easily applied to new instances.)
- The ports for each instance are labeled with a specific SELinux context.
- All Directory Server processes are constrained within the appropriate domain.
- Each domain has specific rules that define what actions that are authorized for the domain.
- Any access not specified in the SELinux policy is denied to the instance.

SELinux has three different levels of enforcement: disabled (no SELinux), permissive (where the rules are processed but not enforced), and enforcing (where all rules are strictly enforced). Red Hat Directory Server has defined SELinux policies that allow it to run as normal under strict SELinux enforcing mode, with a caveat. The Directory Server can run in different modes, one for normal operations and one for database operations like importing (ldif2db mode). The SELinux policies for the Directory Server only apply to normal mode.

By default, the Directory Server runs confined by SELinux policies.

SELinux itself is much more complex to manage and implement than what is described here. This section is concerned only with giving the SELinux details for the Directory Server. Both the Fedora project and the National Security Agency have excellent resources for learning about SELinux. SELinux is a feature of Red Hat Enterprise Linux and, as such, is covered in the Red Hat Enterprise Linux *SELinux Guide* at http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Security-Enhanced_Linux/index.html.

Both the Directory Server and the Admin Server have their own defined SELinux policies. The SELinux policies for the Directory Server and SNMP services are installed in the **389-ds-selinux** RPM package, and the policies for the Admin Server are in the **389-ds-admin-selinux** RPM package.

The policies for each Directory Server instance are updated, if necessary, each time the instance is configured with the setup scripts. The same is true for the Admin Server policies.

9.11. OTHER SECURITY RESOURCES

For more information about designing a secure directory, see the following:

- *Understanding and Deploying LDAP Directory Services.* T. Howes, M. Smith, G. Good, Macmillan Technical Publishing, 1999.
- SecurityFocus.com http://www.securityfocus.com

- Computer Emergency Response Team (CERT) Coordination Center http://www.cert.org

CHAPTER 10. DIRECTORY DESIGN EXAMPLES

The design the directory service depends on the size and nature of the enterprise. This chapter provides a couple of examples of how a directory can be applied within a variety of different settings. These examples are a starting point for developing a real-life directory service deployment plan.

10.1. DESIGN EXAMPLE: A LOCAL ENTERPRISE

Example Corp., an automobile parts manufacturer, is a small company that consists of 500 employees. Example Corp. decides to deploy Red Hat Directory Server to support the directory-enabled applications it uses.

10.1.1. Local Enterprise Data Design

Example Corp. first decides the type of data it will store in the directory. To do this, Example Corp. creates a deployment team that performs a site survey to determine how the directory will be used. The deployment team determines the following:

- Example Corp.'s directory will be used by a messaging server, a web server, a calendar server, a human resources application, and a white pages application.
- The messaging server performs exact searches on attributes such as ***uid***, ***mailServerName***, and ***mailAddress***. To improve database performance, Example Corp. will maintain indexes for these attributes to support searches by the messaging server.

  For more information on using indexes, see Section 6.4, “Using Indexes to Improve Database Performance” .
- The white pages application frequently searches for user names and phone numbers. The directory therefore needs to be capable of frequent substring, wildcard, and fuzzy searches, which return large sets of results. Example Corp. decides to maintain presence, equality, approximate, and substring indexes for the ***cn***, ***sn***, and ***givenName*** attributes and presence, equality, and substring indexes for the ***telephoneNumber*** attribute.
- Example Corp.'s directory maintains user and group information to support an LDAP server-based intranet deployed throughout the organization. Most of Example Corp.'s user and group information will be centrally managed by a group of directory administrators. However, Example Corp. also wants email information to be managed by a separate group of mail administrators.
- Example Corp. plans to support public key infrastructure (PKI) applications in the future, such as S/MIME email, so it needs to be prepared to store users' public key certificates in the directory.

10.1.2. Local Enterprise Schema Design

Example Corp.'s deployment team decides to use the **inetOrgPerson** object class to represent the entries in the directory. This object class is appealing because it allows the ***userCertificate*** and ***uid*** (userID) attributes, both of which are needed by the applications supported by Example Corp.'s directory.

Example Corp. also wants to customize the default directory schema. Example Corp. creates the **examplePerson** object class to represent employees of Example Corp. It derives this object class from the **inetOrgPerson** object class.

The **examplePerson** object class allows one attribute, the ***exampleID*** attribute. This attribute contains the special employee number assigned to each Example Corp. employee.

In the future, Example Corp. can add new attributes to the **examplePerson** object class as needed.

10.1.3. Local Enterprise Directory Tree Design

Based on the data and schema design described in the preceding sections, Example Corp. creates the following directory tree:

- The root of the directory tree is Example Corp.'s Internet domain name: ***dc=example,dc=com***.
- The directory tree has four branch points: ***ou=people***, ***ou=groups***, ***ou=roles,*** and ***ou=resources***.
- All of Example Corp.'s people entries are created under the ***ou=people*** branch.

  The people entries are all members of the ***person***, ***organizationalPerson***, **inetOrgPerson**, and **examplePerson** object classes. The ***uid*** attribute uniquely identifies each entry's DN. For example, Example Corp. contains entries for Babs Jensen (***uid=bjensen)*** and Emily Stanton (***uid=estanton***).

- They create three roles, one for each department in Example Corp.: sales, marketing, and accounting.

  Each person entry contains a role attribute which identifies the department to which the person belongs. Example Corp. can now create ACIs based on these roles.

  For more information about roles, see Section 4.3.2, "About Roles".

- They create two group branches under the ***ou=groups*** branch.

  The first group, ***cn=administrators***, contains entries for the directory administrators, who manage the directory contents.

  The second group, ***cn=messaging admin***, contains entries for the mail administrators, who manage mail accounts. This group corresponds to the administrators group used by the messaging server. Example Corp. ensures that the group it configures for the messaging server is different from the group it creates for Directory Server.

- They create two branches under the ***ou=resources*** branch, one for conference rooms (***ou=conference rooms***) and one for offices (***ou=offices***).

- They create a *class of service (CoS)* that provides values for the ***mailquota*** attribute depending on whether an entry belongs to the administrative group.

  This CoS gives administrators a mail quota of 100GB while ordinary Example Corp. employees have a mail quota of 5GB.

  See Section 5.3, "About Classes of Service" for more information about class of service.

The following diagram illustrates the directory tree resulting from the design steps listed above:

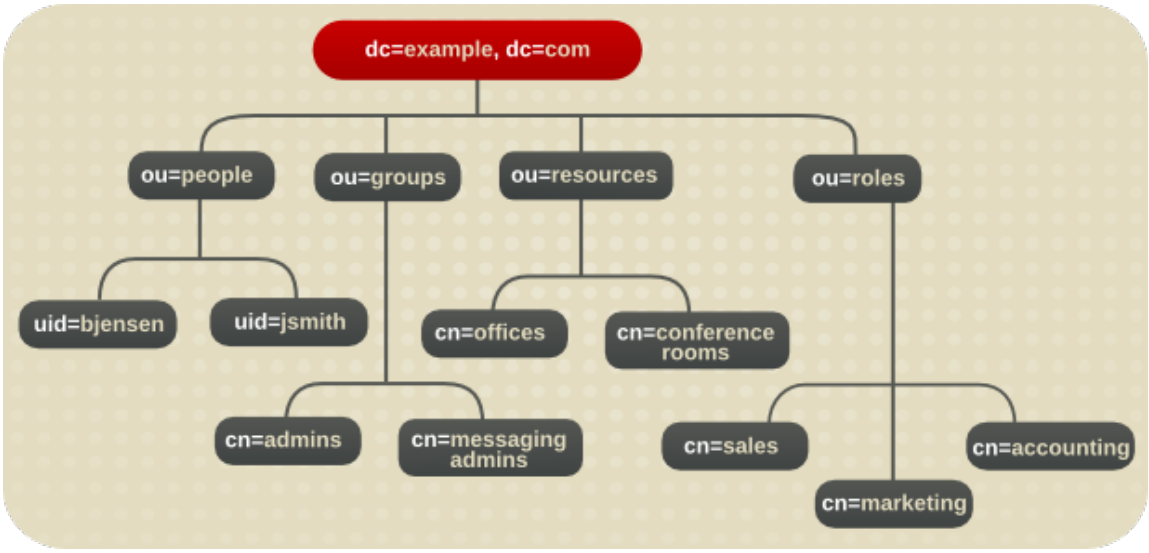


Figure 10.1. Directory Tree for Example Corp.

10.1.4. Local Enterprise Topology Design

At this point, Example Corp. needs to design its database and server topologies. The following sections describe each topology in detail.

10.1.4.1. Database Topology

The company designs a database topology in which the people branch is stored in one database (DB1), the groups branch is stored in another database (DB2), and the resources branch, roles branch, and the ***root suffix*** information are stored in a third database (DB3). This is illustrated in Figure 10.2, "Database Topology for Example Corp." .

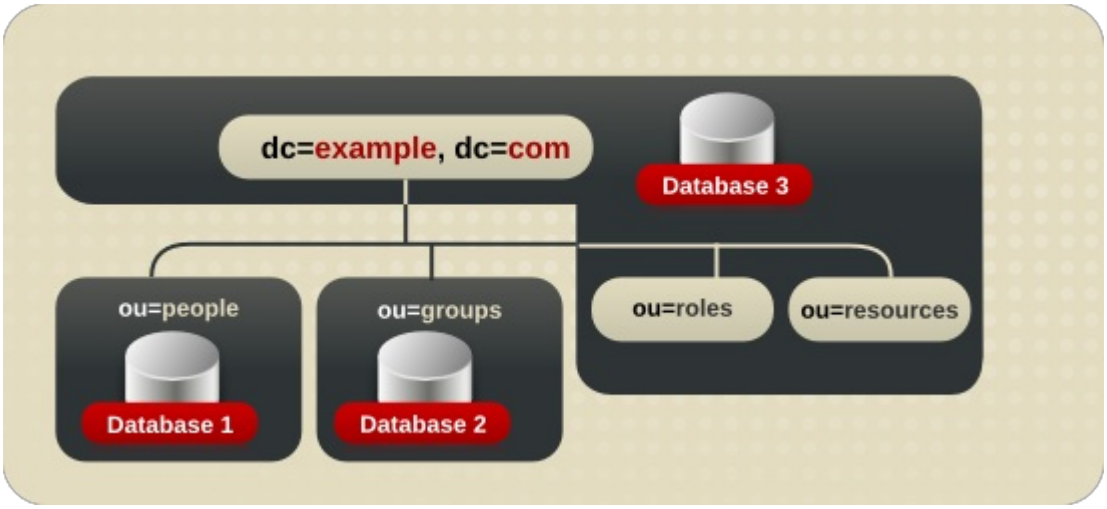


Figure 10.2. Database Topology for Example Corp.

Each of the two supplier servers updates all three consumer servers in Example Corp.'s deployment of Directory Server. These consumers supply data to one messaging server and to the other unified user management products.

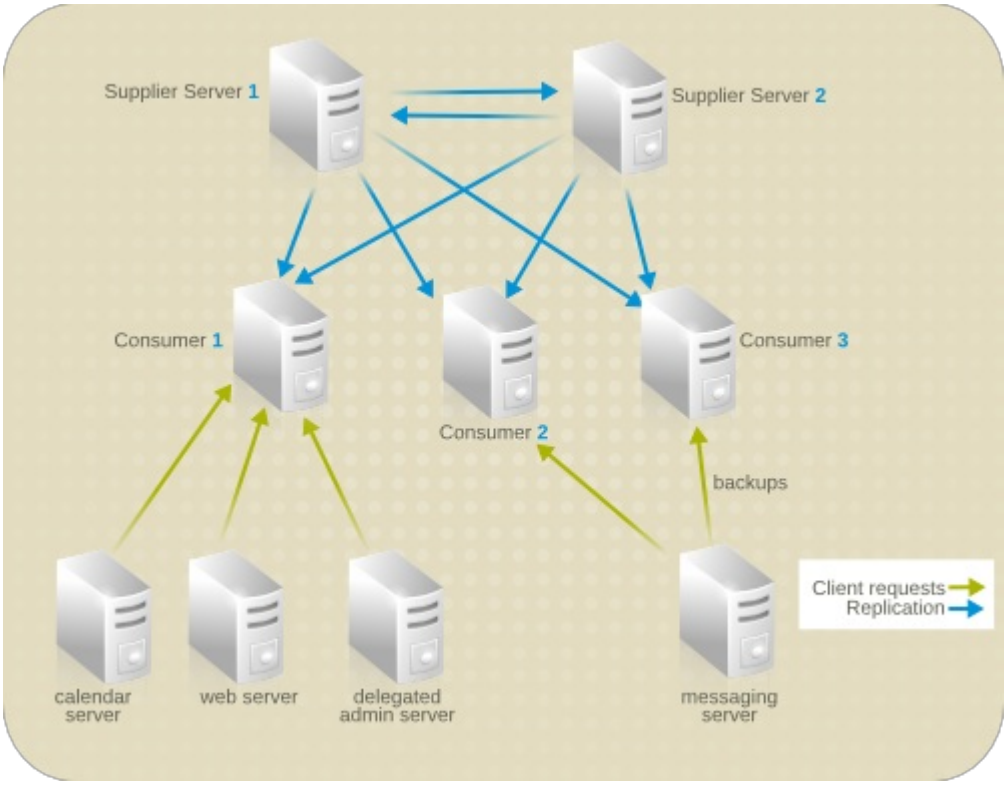


Figure 10.3. Server Topology for Example Corp.

Modify requests from compatible servers are routed to the appropriate consumer server. The consumer server uses smart referrals to route the request to the supplier server responsible for the master copy of the data being modified.

10.1.5. Local Enterprise Replication Design

Example Corp. decides to use a multi-master replication design to ensure the high availability of its directory data. For more information about multi-master replication, see Section 7.2.2, "Multi-Master Replication".

The following sections provide more details about the supplier server architecture and the supplier-consumer server topology.

10.1.5.1. Supplier Architecture

Example Corp. uses two supplier servers in a multi-master replication architecture. The suppliers update one another so that the directory data remains consistent. The supplier architecture for Example Corp. is described below:

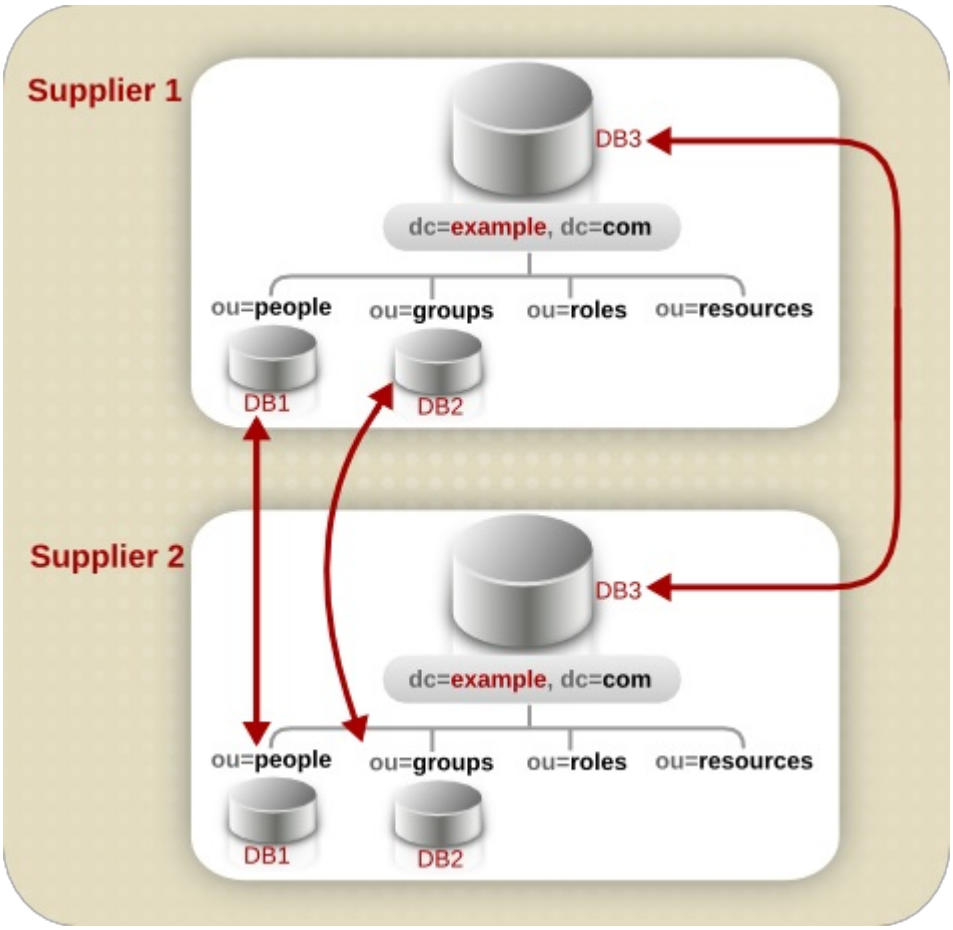


Figure 10.4. Supplier Architecture for Example Corp.

10.1.5.2. Supplier Consumer Architecture

The following diagram describes how the supplier servers replicate to each consumer in the Example Corp. deployment of the directory. Each of the three consumer servers is updated by the two supplier servers. This ensures that the consumers will not be affected if there is a failure in one of the supplier servers.

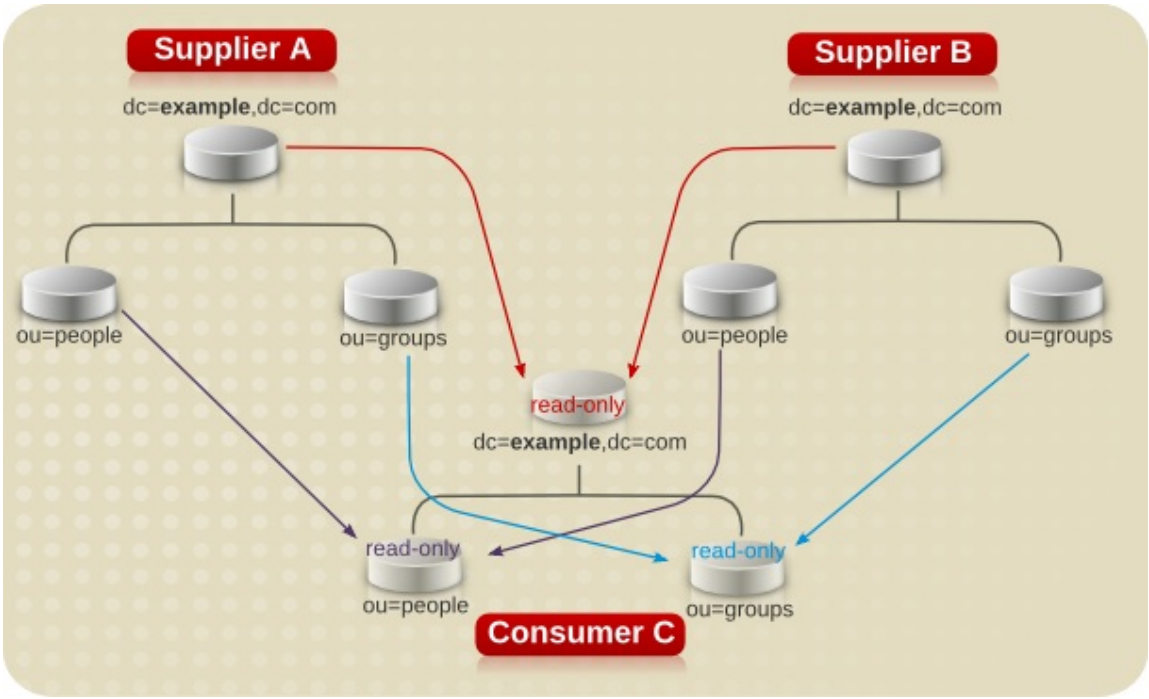


Figure 10.5. Supplier and Consumer Architecture for Example Corp.

10.1.6. Local Enterprise Security Design

Example Corp. decides on the following security design to protect its directory data:

- They create an ACI that allows employees to modify their own entries.

  Users can modify all attributes except the ***uid***, ***manager*** and ***department*** attributes.

- To protect the privacy of employee data, they create an ACI that allows only the employee and their manager to see the employee's home address and phone number.

- They create an ACI at the root of the directory tree that allows the two administrator groups the appropriate directory permissions.

  The directory administrators group needs full access to the directory. The messaging administrators group needs write and delete access to the **mailRecipient** and **mailGroup** object classes and the attributes contained on those object classes, as well as the ***mail*** attribute. Example Corp. also grants the messaging administrators group ***write***, ***delete***, and ***add*** permissions to the group subdirectory for creation of mail groups.

- They create a general ACI at the root of the directory tree that allows anonymous access for read, search, and compare access.

  This ACI denies anonymous write access to password information.

- To protect the server from denial of service attacks and inappropriate use, they set resource limits based on the ***DN*** used by directory clients to bind.

  Example Corp. allows anonymous users to receive 100 entries at a time in response to search requests, messaging administrative users to receive 1,000 entries, and directory administrators to receive an unlimited number of entries.

  For more information about setting resource limits based on the bind DN, see the "User Account Management" chapter in the *Red Hat Directory Server Administrator's Guide* .

- They create a password policy which specifies that passwords must be at least eight characters in length and expire after 90 days.

  For more information about password policies, see Section 9.6, “Designing a Password Policy” .

- They create an ACI that gives members of the accounting role access to all payroll information.

10.1.7. Local Enterprise Tuning and Optimizations

The company uses the following tactics to optimize its deployment of Directory Server:

- Running the **dsktune** utility.

  The **dsktune** utility provides an easy and reliable way of checking the patch levels and kernel parameter settings of the system. For more information about **dsktune**, see the *Red Hat Directory Server Installation Guide*.

- Optimizing the entry and database caches.

  Example Corp. sets the entry cache size to 2GB entries and the database cache to 250MB to ensure that all of the indexes fit into RAM, optimizing server performance.

10.1.8. Local Enterprise Operations Decisions

The company makes the following decisions regarding the day-to-day operation of its directory:

- Back up the databases every night.

- Use SNMP to monitor the server status.

  For more information about SNMP, see the *Red Hat Directory Server Administrator's Guide* .

- Auto-rotate the access and error logs.

- Monitor the error log to ensure that the server is performing as expected.

- Monitor the access log to screen for searches that should be indexed.

For more information about the access, error, and audit logs, see the "Monitoring Server and Database Activity" chapter in the *Red Hat Directory Server Administrator's Guide* .

10.2. DESIGN EXAMPLE: A MULTINATIONAL ENTERPRISE AND ITS EXTRANET

This example builds a directory infrastructure for Example Corp. International. The Example Corp. from the previous example has grown into a large, multinational company. This example builds on the directory structure created in the last example for Example Corp., expanding the directory design to meet its new needs.

Example Corp. has grown into an organization dispersed over three main geographic locations: the US, Europe, and Asia. Example Corp. now has more than 20,000 employees, all of whom live and work in the countries where the Example Corp. offices are located. Example Corp. decides to launch a company-wide LDAP directory to improve internal communication, to make it easier to develop and deploy web applications, and to increase security and privacy.

Designing a directory tree for an international corporation involves determining how to collect directory entries logically, how to support data management, and how to support replication on a global scale.

In addition, Example Corp. wants to create an extranet for use by its parts suppliers and trading partners. An *extranet* is an extension of an enterprise's intranet to external clients.

The following sections describe the steps in the process of deploying a multinational directory service and extranet for Example Corp. International.

10.2.1. Multinational Enterprise Data Design

Example Corp. International creates a deployment team to perform a site survey. The deployment team determines the following from the site survey:

- A messaging server is used to provide email routing, delivery, and reading services for most of Example Corp.'s sites. An enterprise server provides document publishing services. All servers run on Red Hat Enterprise Linux 5 (64-bit).
- Example Corp. needs to allow data to be managed locally. For example, the European site will be responsible for managing the Europe branch of the directory. This also means that Europe will be responsible for the master copy of its data.
- Because of the geographic distribution of Example Corp.'s offices, the directory needs to be available to users and applications 24 hours a day.
- Many of the data elements need to accommodate data values of several different languages.

NOTE

All data use the UTF-8 characterset; any other characterset violates LDAP standards.

The deployment team also determines the following about the data design of the extranet:

- Parts suppliers need to log in to Example Corp.'s directory to manage their contracts with Example Corp. Parts suppliers depend on data elements used for authentication, such as name and user password.
- Example Corp.'s partners will use the directory to look up contact details of people in the partner network, such as email addresses and phone numbers.

10.2.2. Multinational Enterprise Schema Design

Example Corp. builds upon its original schema design by adding schema elements to support the extranet. Example Corp. adds two new objects, the **exampleSupplier** object class and the **examplePartner** object class.

The **exampleSupplier** object class allows one attribute, the ***exampleSupplierID*** attribute. This attribute contains the unique ID assigned by Example Corp. International to each automobile parts supplier with which it works.

The **examplePartner** object class allows one attribute, the ***examplePartnerID*** attribute. This attribute contains the unique ID assigned by Example Corp. International to each trade partner.

For information about customizing the default directory schema, see Section 3.4, “Customizing the Schema”.

10.2.3. Multinational Enterprise Directory Tree Design

Based on the expanded requirements, Example Corp. creates the following directory tree:

- The root of the directory tree is the ***dc=com*** suffix. Under this suffix, Example Corp. creates two branches. One branch, ***dc=exampleCorp,dc=com***, contains data internal to Example Corp. International. The other branch, ***dc=exampleNet,dc=com***, contains data for the extranet.
- The directory tree for the intranet (under ***dc=exampleCorp,dc=com)*** has three main branches, each corresponding to one of the regions where Example Corp. has offices. These branches are identified using the ***l*** (locality) attribute.
- Each main branch under ***dc=exampleCorp,dc=com*** mimics the original directory tree design of Example Corp. Under each locality, Example Corp. creates an ***ou=people***, an ***ou=groups***, an ***ou=roles***, and an ***ou=resources*** branch. See Figure 10.1, “Directory Tree for Example Corp.” for more information about this directory tree design.
- Under the ***dc=exampleNet,dc=com*** branch, Example Corp. creates three branches. One branch for suppliers (***o=suppliers***), one branch for partners (***o=partners***), and one branch for groups (***ou=groups***).
- The ***ou=groups*** branch of the extranet contains entries for the administrators of the extranet as well as for mailing lists that partners subscribe to for up-to-date information on automobile parts manufacturing.

The following diagram illustrates the basic directory tree resulting from the design steps listed above:

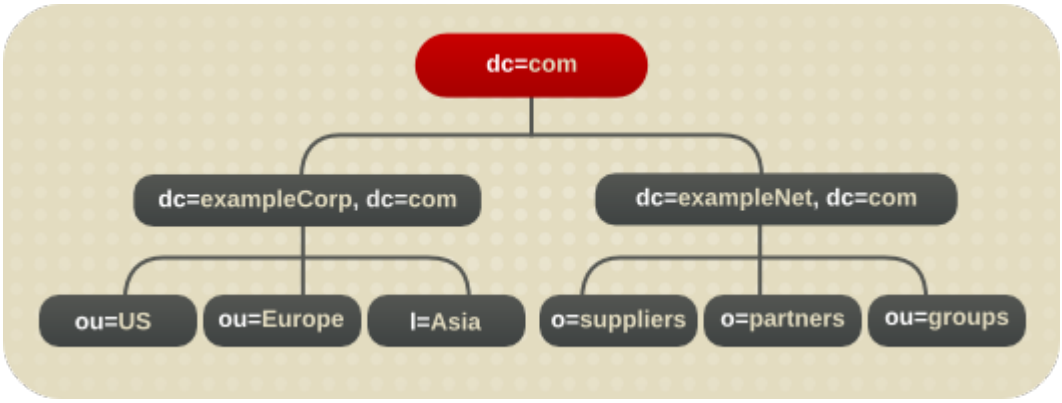


Figure 10.6. Basic Directory Tree for Example Corp. International

The following diagram illustrates the directory tree for the Example Corp. intranet:

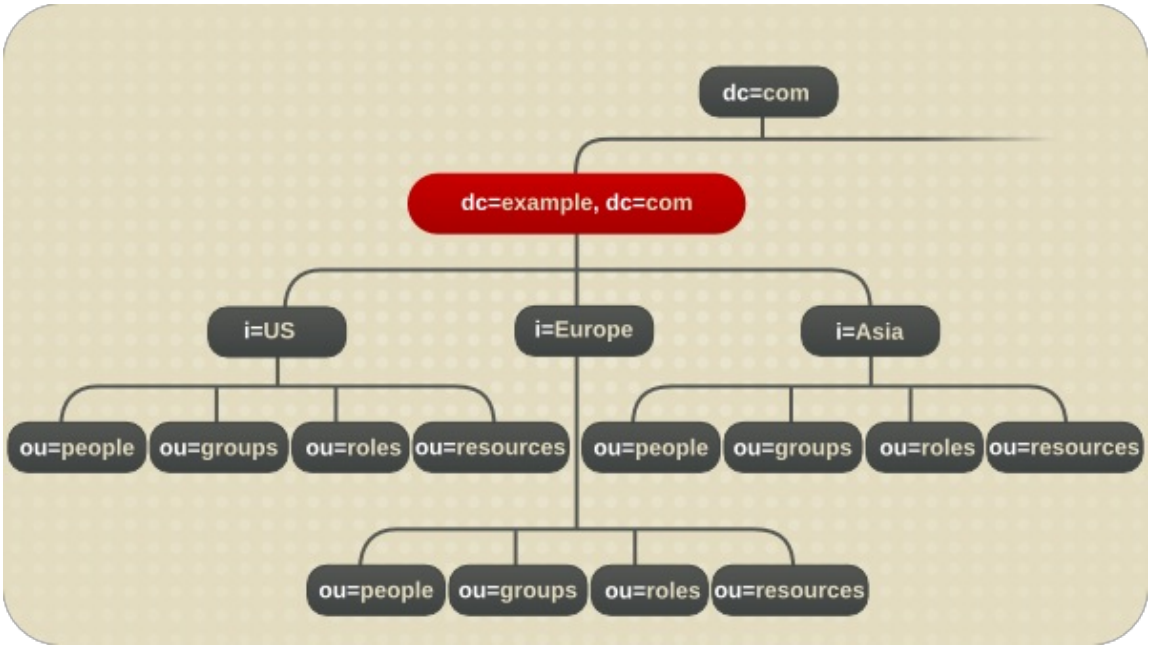


Figure 10.7. Directory Tree for Example Corp. International's Intranet

The entry for the ***l=Asia*** entry appears in LDIF as follows:

```
dn: l=Asia,dc=exampleCorp,dc=com

objectclass: top
```

```
objectclass: locality

l: Asia

description: includes all sites in Asia
```

The following diagram illustrates the directory tree for Example Corp.'s extranet:

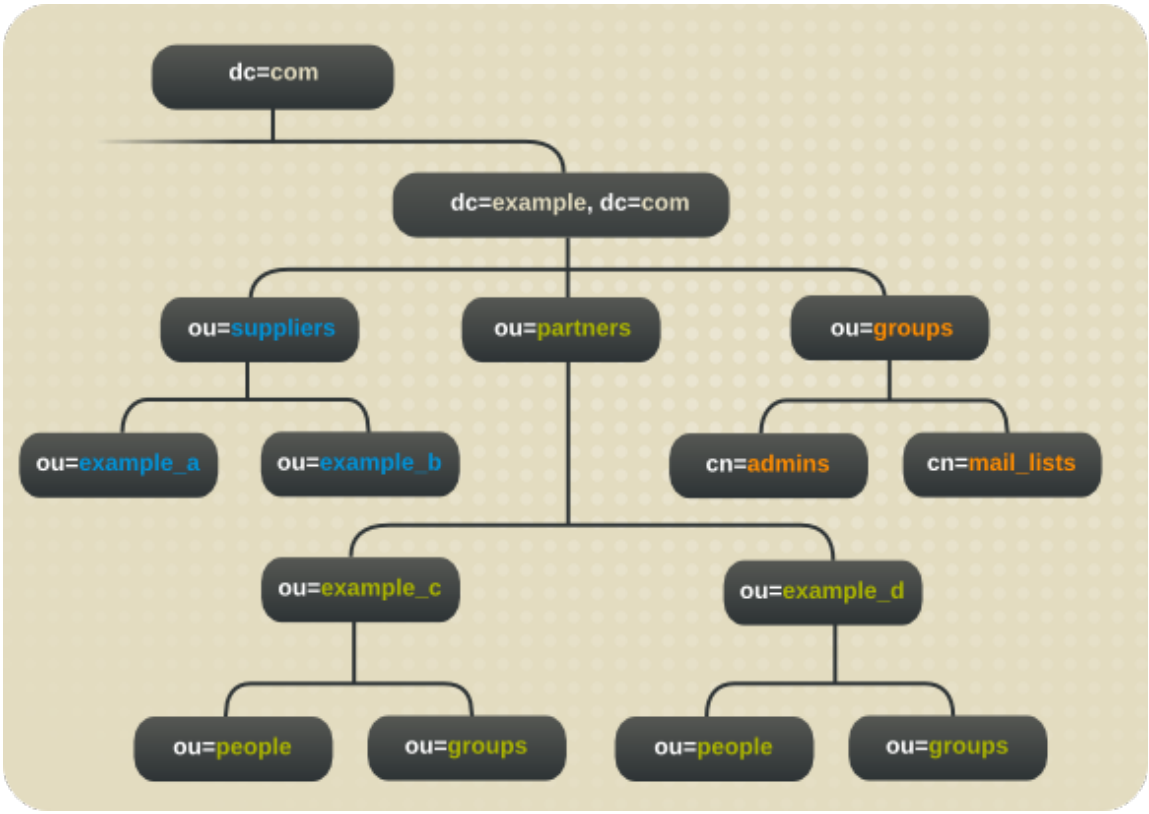


Figure 10.8. Directory Tree for Example Corp. International's Extranet

10.2.4. Multinational Enterprise Topology Design

At this point, Example Corp. designs its database and server topologies. The following sections describe each topology in more detail.

10.2.4.1. Database Topology

The following diagram illustrates the database topology of one of Example Corp.'s main localities, Europe:

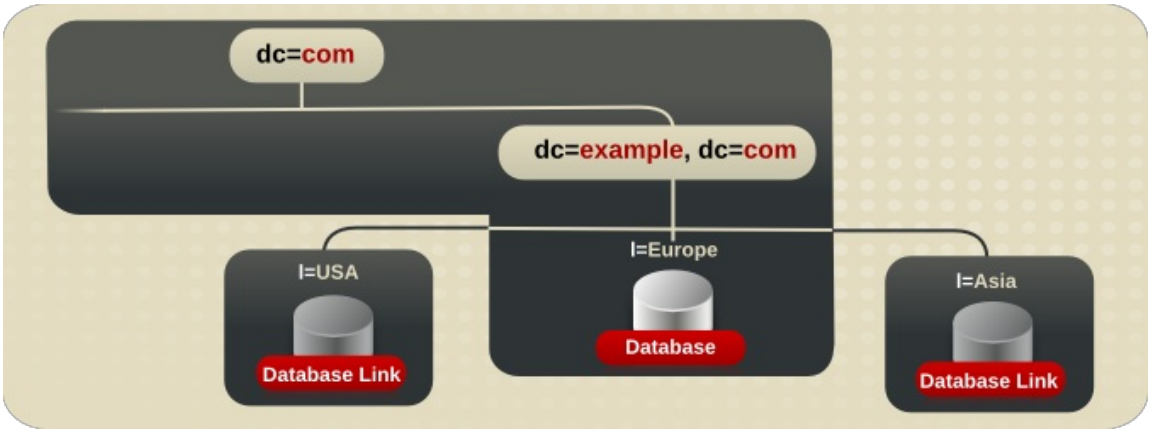


Figure 10.9. Database Topology for Example Corp. Europe

The database links point to databases stored locally in each country. For example, operation requests received by the Example Corp. Europe server for the data under the ***l=US*** branch are chained by a database link to a database on a server in Austin, Texas. For more information about database links and chaining, see Section 6.3.2, “Using Chaining”.

The master copy of the data for ***dc=exampleCorp,dc=com*** and the root entry, ***dc=com***, is stored in the ***l=Europe*** database.

The data center in Europe contains the master copies of the data for the extranet. The extranet data is stored in three databases, one for each of the main branches. The master copy of the data for ***o=suppliers*** is stored in database one (DB1), that for ***o=partners*** is stored in database two (DB2), and that for ***ou=groups*** is stored in database three (DB3).

The database topology for the extranet is illustrated below:

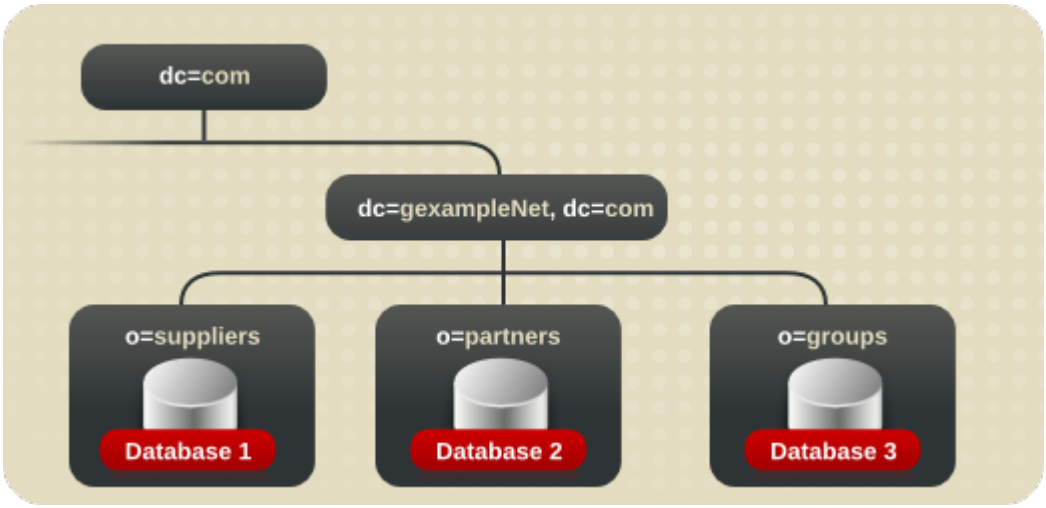


Figure 10.10. Database Topology for Example Corp. International's Extranet

10.2.4.2. Server Topology

Example Corp. develops two server topologies, one for the corporate intranet and one for the partner extranet.

For the intranet, Example Corp. decides to have a master database for each major locality. This means it has three data centers, each containing two supplier servers, two hub servers, and three consumer servers.

The following diagram illustrates the architecture of Example Corp. Europe's data center:

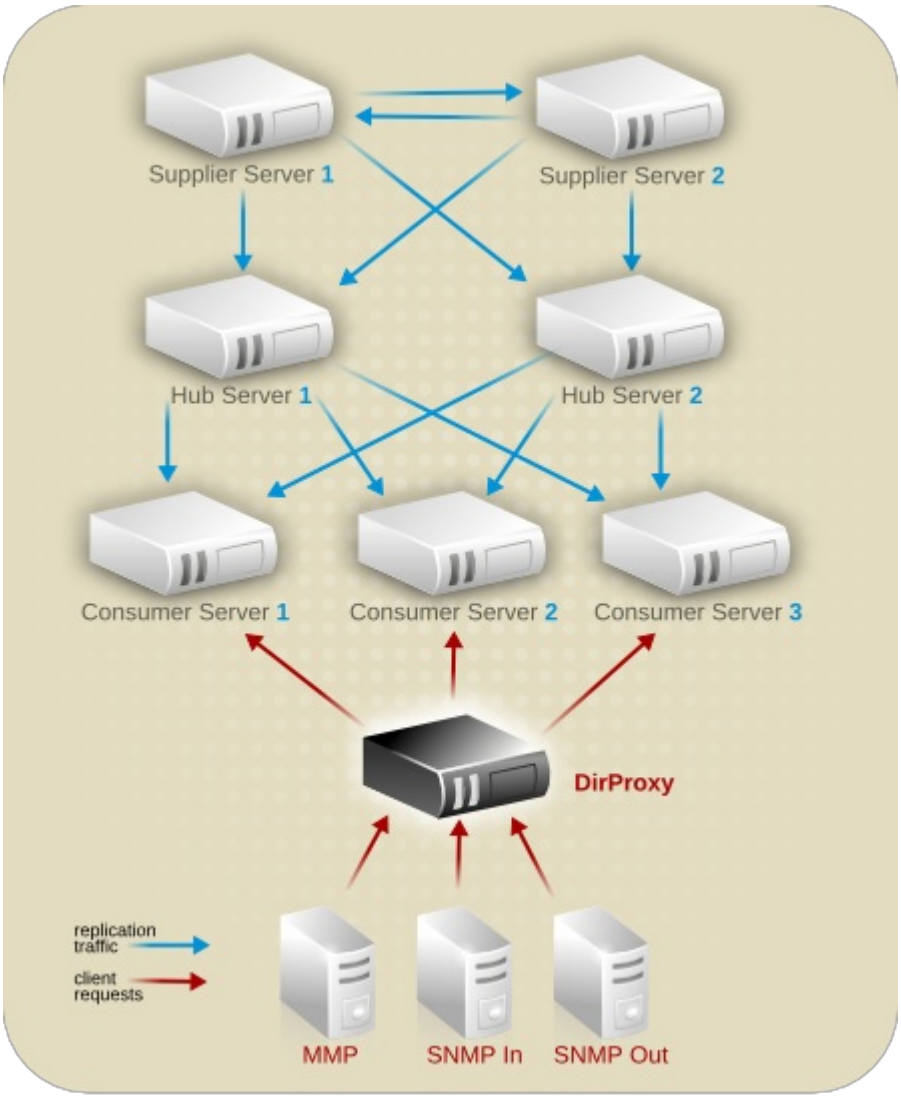


Figure 10.11. Server Topology for Example Corp. Europe

The data master for Example Corp.'s extranet is in Europe. This data is replicated to two consumer servers in the US data center and two consumer servers in the Asia data center. Overall, Example Corp. requires ten servers to support the extranet.

The following diagram illustrates the server architecture of Example Corp.'s extranet in the European data center:

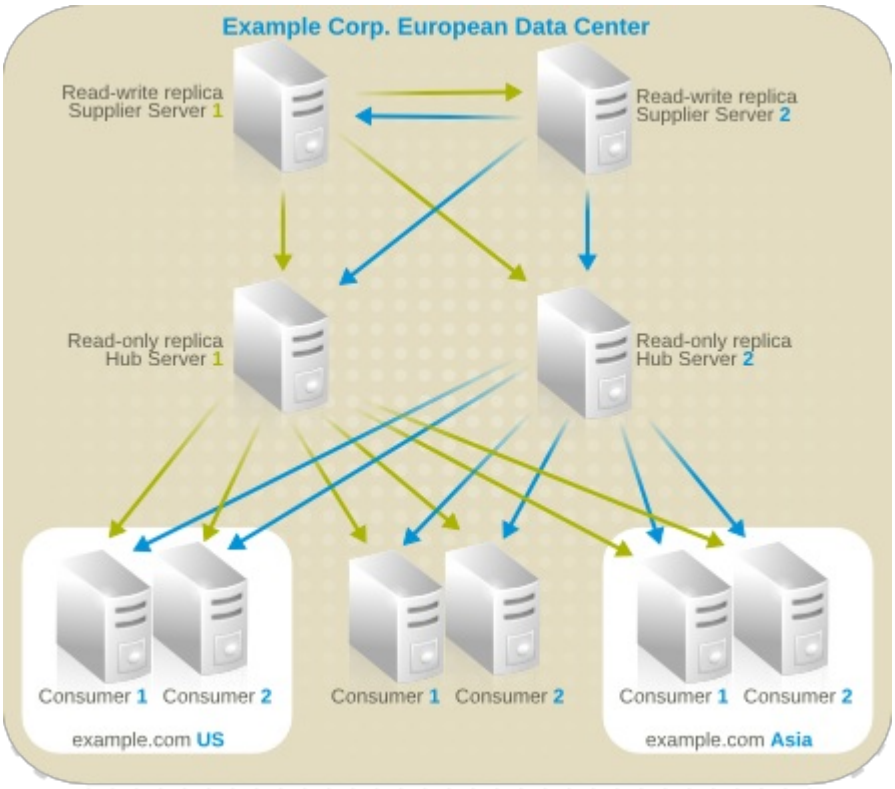


Figure 10.12. Server Topology for Example Corp. International's Extranet

The hub servers replicate data to two consumer servers in each of the data centers in Europe, the US and Asia.

10.2.5. Multinational Enterprise Replication Design

Example Corp. considers the following points when designing replication for its directory:

- Data will be managed locally.
- The quality of network connections varies from site to site.
- Database links will be used to connect data on remote servers.
- Hub servers that contain read-only copies of the data will be used to replicate data to consumer servers.

The hub servers are located near important directory-enabled applications such as a mail server or a web server.

Hub servers remove the burden of replication from the supplier servers, so the supplier servers can focus on write operations. In the future, as Example Corp. expands and needs to add more consumer servers, the additional consumers do not affect the performance of the supplier servers.

For more information on hub servers, see Section 7.2.3, “Cascading Replication”.

10.2.5.1. Supplier Architecture

For the Example Corp. intranet, each locality stores the master copy of its data and uses database links to chain to the data of other localities. For the master copy of its data, each locality uses a multi-master replication architecture.

The following diagram illustrates the supplier architecture for Europe, which includes the ***dc=exampleCorp,dc=com*** and ***dc=com*** information:

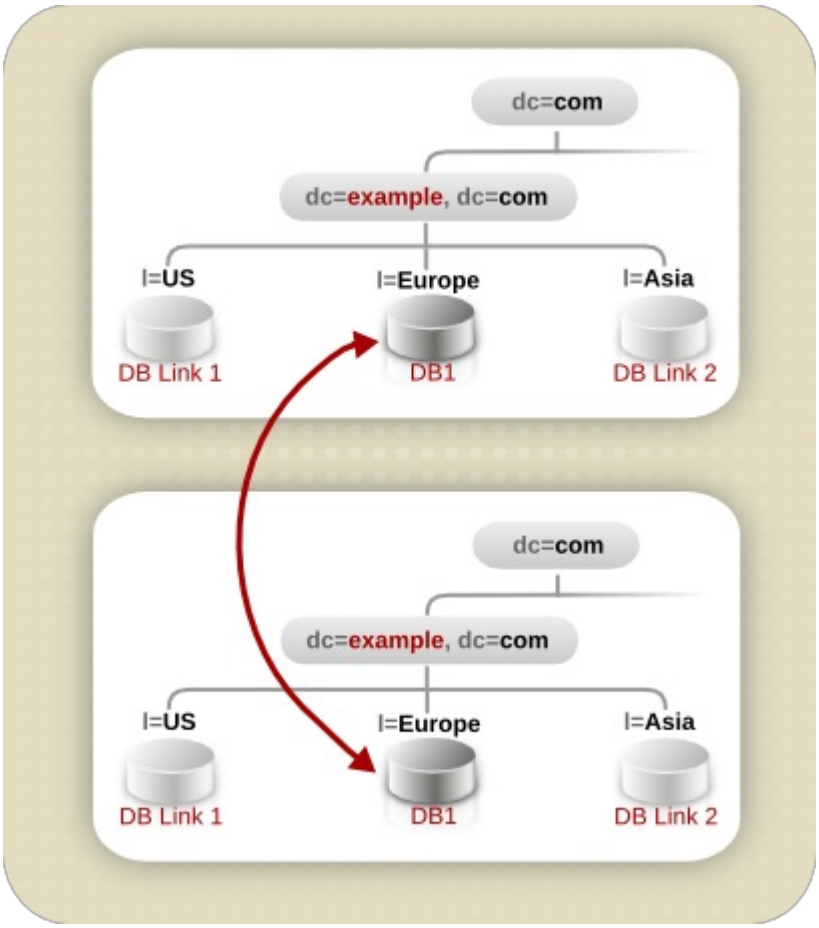


Figure 10.13. Supplier Architecture for Example Corp. Europe

Each locality contains two suppliers, which share master copies of the data for that site. Each locality is therefore responsible for the master copy of its own data. Using a multi-master architecture ensures the availability of the data and helps balance the workload managed by each supplier server.

To reduce the risk of total failure, Example Corp. uses multiple read-write supplier Directory Servers at each site.

The following diagram illustrates the interaction between the two supplier servers in Europe and the two supplier servers in the US:

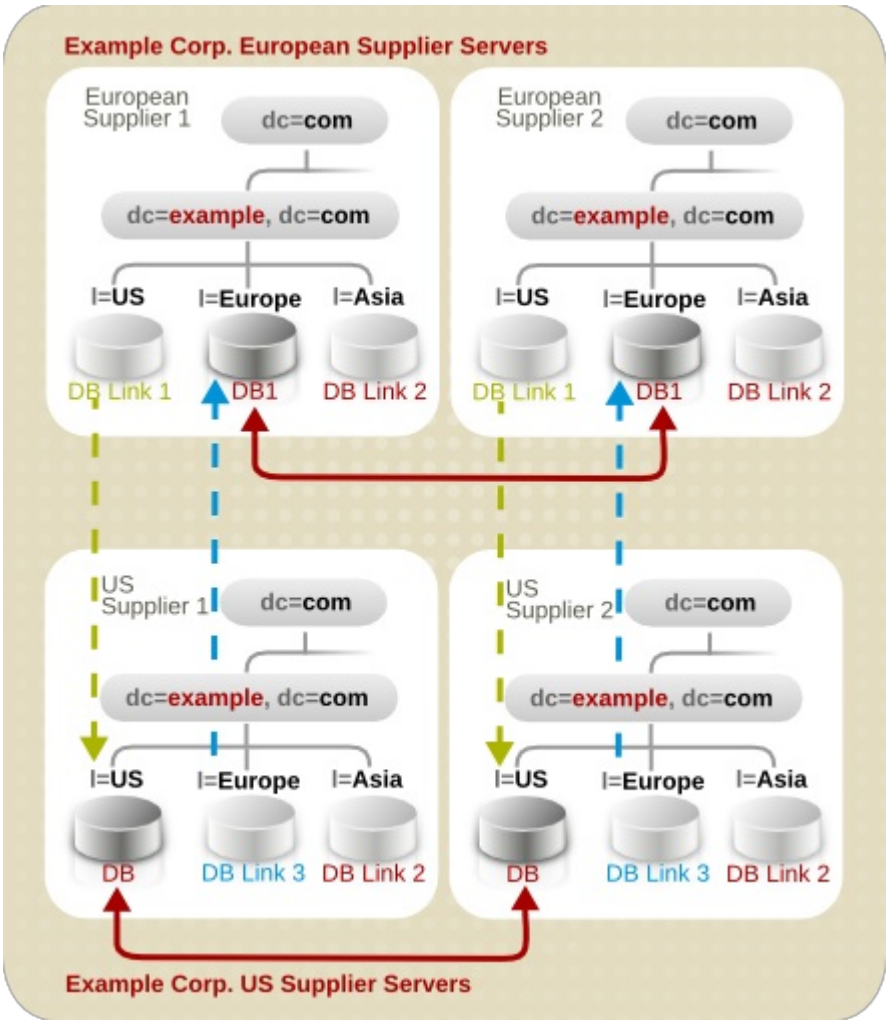


Figure 10.14. Multi-Master Replication Design for Example Corp. Europe and Example Corp. US

The same relationship exists between Example Corp. US and Example Corp. Asia, and between Example Corp. Europe and Example Corp. Asia.

10.2.6. Multinational Enterprise Security Design

Example Corp. International builds upon its previous security design, adding the following access controls to support its new multinational intranet:

- Example Corp. adds general ACIs to the root of the intranet, creating more restrictive ACIs in each country and the branches beneath each country.
- Example Corp. decides to use macro ACIs to minimize the number of ACIs in the directory.

  Example Corp. uses a macro to represent a DN in the target or bind rule portion of the ACI. When the directory gets an incoming LDAP operation, the ACI macros are matched against the resource targeted by the LDAP operation. If there is a match, the macro is replaced by the value of the DN of the targeted resource.

  For more information about macro ACIs, see the *Red Hat Directory Server Administrator's Guide* .

Example Corp. adds the following access controls to support its extranet:

- Example Corp. decides to use certificate-based authentication for all extranet activities. When people log in to the extranet, they need a digital certificate. The directory is used to store the certificates. Because the directory stores the certificates, users can send encrypted email by looking up public keys stored in the directory.
- Example Corp. creates an ACI that forbids anonymous access to the extranet. This protects the extranet from denial of service attacks.
- Example Corp. wants updates to the directory data to come only from an Example Corp. hosted application. This means that partners and suppliers using the extranet can only use the tools provided by Example Corp. Restricting extranet users to Example Corp.'s preferred tools allows Example Corp. administrators to use the

audit logs to track the use of the directory and limits the types of problems that can be introduced by extranet users outside of Example Corp. International.

APPENDIX A. DIRECTORY SERVER RFC SUPPORT

NOTE

This chapter lists notable supported LDAP-related RFCs. It is not a complete list of RFCs Directory Server supports.

A.1. LDAPV3 FEATURES

Technical Specification Road Map (RFC 4510)

This is a tracking document and does not contain requirements.

The Protocol (RFC 4511)

Supported. Exceptions:

- RFC 4511 Section 4.4.1. Notice of Disconnection : Directory Server terminates the connections in this case.
- RFC 4511 Section 4.5.1.3. SearchRequest.derefAliases : LDAP aliases are not supported.
- RFC 4511 Section 4.13. IntermediateResponse Message

Directory Information Models (RFC 4512)

Supported. Exceptions:

- RFC 4512 Section 2.4.2. Structural Object Classes : Directory Server supports entries with multiple structural object classes.
- RFC 4512 Section 2.6. Alias Entries
- RFC 4512 Section 4.1.2. Attribute Types : The attribute type ***COLLECTIVE*** is not supported.
- RFC 4512 Section 4.1.4. Matching Rule Uses
- RFC 4512 Section 4.1.6. DIT Content Rules
- RFC 4512 Section 4.1.7. DIT Structure Rules and Name Forms
- RFC 4512 Section 5.1.1. altServer

Note that RFC 4512 enables LDAP servers to not support the previously listed exceptions. For further details, see RFC 4512 Section 7.1. Server Guidelines .

Authentication Methods and Security Mechanisms (RFC 4513)

Supported.

String Representation of Distinguished Names (RFC 4514)

Supported.

String Representation of Search Filters (RFC 4515)

Supported.

Uniform Resource Locator (RFC 4516)

Supported. However, this RFC is mainly focused on LDAP clients.

Syntaxes and Matching Rules (RFC 4517)

Supported. Exceptions:

- ***directoryStringFirstComponentMatch***
- ***integerFirstComponentMatch***
- ***objectIdentifierFirstComponentMatch***
- ***objectIdentifierFirstComponentMatch***
- ***keywordMatch***

- ***wordMatch***

Internationalized String Preparation (RFC 4518)

Supported.

Schema for User Applications (RFC 4519)

Supported.

A.2. AUTHENTICATION METHODS

Anonymous SASL Mechanism (RFC 4505)

Not supported. Note that RFC 4512 does not require the **ANONYMOUS** SASL mechanism. However, Directory Server support LDAP anonymous binds.

External SASL Mechanism (RFC 4422)

Supported.

Plain SASL Mechanism (RFC 4616)

Not supported. Note that RFC 4512 does not require the **PLAIN** SASL mechanism. However, Directory Server support LDAP anonymous binds.

SecurID SASL Mechanism (RFC 2808)

Not supported. However if a Cyrus SASL plug-in exists, Directory Server is able to use it.

Kerberos V5 (GSSAPI) SASL Mechanism (RFC 4752)

Supported.

CRAM-MD5 SASL Mechanism (RFC 2195)

Supported.

Digest-MD5 SASL Mechanism (RFC 2831)

Supported.

One-time Password SASL Mechanism (RFC 2444)

Not supported. However if a Cyrus SASL plug-in exists, Directory Server is able to use it.

A.3. X.509 CERTIFICATES SCHEMA AND ATTRIBUTES SUPPORT

LDAP Schema Definitions for X.509 Certificates (RFC 4523)

- Attribute types and object classes: Supported.
- Syntaxes: Not supported. Directory Server uses binary and octet syntax.
- Matching rules: Not supported.

CHAPTER 11. REVISION HISTORY

Note that revision numbers relate to the edition of this manual, not to version numbers of Red Hat Directory Server.

Revision 9.1-11 **June 26, 2017** **Marc Muehlfeld**
Added a statement that this documentation is deprecated and no longer maintained.

Revision 9.1-10 **February 24, 2017** **Marc Muehlfeld**
Added "Directory Server RFC Support" appendix.

Revision 9.1-9 **December 16, 2016** **Marc Muehlfeld**
Minor update in the About the ACI Format section.

Revision 9.1-2 **February 21, 2013** **Ella Deon Lackey**
Updates for RHEL 6.4.

Revision 9.0-2 **July 2, 2012** **Ella Deon Lackey**
Added Automembership Plug-in information.

Revision 9.0-1 **January 30, 2012** **Ella Deon Lackey**
Added information for the Account Policy Plug-in.

Revision 9.0-0 **December 6, 2011** **Ella Deon Lackey**
Initial version for Directory Server version 9.0.