Novell Developer Kit

www.novell.com

March 1, 2006

NOVELL® EDIRECTORY™ TECHNICAL OVERVIEW

Legal Notices

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell developer products, and to get updates, see developer.novell.com/ndk. To access online documentation for Novell products, see www.novell.com/documentation.

Novell Trademarks

Third-Party Materials

Contents

Preface

Novell® eDirectory™ is a highly scalable, secure directory service. This book provides an overview of Novell eDirectory and its architecture and services. It is divided into the following sections:

Audience

This guide is intended for the developers to understand the Novell’s LDAP server and eDirectory.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation.

Documentation Updates

For the most recent version of this guide, see eDirectory Libraries for C (http://developer.novell.com/ndk/ndslib.htm).

Additional Information

For information about other eDirectory interfaces, see the following guides:

- eDirectory Iterator Services (http://developer.novell.com/ndk/doc/ndslib/skds_enu/data/front.html)
- eDirectory Event Services (http://developer.novell.com/ndk/doc/ndslib/dsev_enu/data/hmwiqbwd.html)
- eDirectory Technical Overview (http://developer.novell.com/ndk/doc/ndslib/dsov_enu/data/h6tvg4z7.html)
- eDirectory Core Services (http://developer.novell.com/ndk/doc/ndslib/nds__enu/data/h2y7hdit.html)
- eDirectory Schema Reference (http://developer.novell.com/ndk/doc/ndslib/schm_enu/data/h4q1mn1i.html)

For help with eDirectory problems or questions, visit the eDirectory Libraries for C Developer Support Forum (http://developer.novell.com/ndk/devforums.htm).

For product information about eDirectory, see the eDirectory Documentation Site (http://www.novell.com/documentation/edirectory.html).

Documentation Conventions

In this documentation, a greater-than symbol (>) is used to separate actions within a step and items within a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux* and UNIX*, should use forward slashes as required by your software.

Novell eDirectory

1

A directory is a database, designed especially to handle the needs of users who need access to network resources. Novell® eDirectory™ is a full-service, platform independent directory that complies with X.500 and LDAP industry diectory standards.

Novell eDirectory is a full service directory that complies with X.500 and LDAP industry directory standards. The following sections provide an overview of these characteristics:

The other chapters in this book provide an in depth description of the various features of eDirectory and NDS® (Novell Directory Services™): object management, security, and partition and replica management. The schema that controls the type of data that can be stored in the database is described in a separate book, *NDK: Novell eDirectory Schema Reference*. Additional information on LDAP and eDirectory is contained in the "LDAP and eDirectory Integration" book. Information on configuring eDirectory is contained in eDirectory Documentation (http://www.novell.com/documentation/edirectory.html).

1.1 NDS and Novell eDirectory

NDS has evolved from a NOS-based, NetWare only directory service into a cross-platform, scalable, standards-based directory called Novell eDirectory. While NDS is available for NetWare, eDirectory is available on NetWare® 6.5, Netware 6, NetWare 5.x, Windows* NT/2000, Solaris*, Linux*, AIX*, and HP-UX*.

The majority of the conceptual information in this book applies to both eDirectory and NDS. Where there are differences, specific versions of NDS or eDirectory are used.

1.2 Deployed Versions of NDS and Novell eDirectory

Table 1-1 *Deployed Versions of NDS and Novell eDirectory*

Product Version	Build Version	Platforms
NetWare 5.1 SP4 (NDS 7)	DS.nlm v7.57	NetWare 5.1
NetWare 5.1 SP 4 (NDS 8)	DS.nlm v8.79	NetWare 5.1

Product Version	Build Version	Platforms
eDirectory 8	DS.nlm & DS.dlm v8.79	NetWare 5.0, Windows NT/2000
eDirectory 8.5.x	DS v85.23	NetWare 5.x, Windows, Solaris
NetWare 6 (eDirectory 8.6)	DS.nlm v10110.20	NetWare 6
eDirectory 8.6.1	DS v10210.43	NetWare 6, NetWare 5.1, Windows, Solaris, Linux
NetWare 6 SP1 (eDirectory 8.6.2)	DS.nlm v10310.17	NetWare 6
eDirectory 8.6.2	DS v103xx.xx	NetWare 6, NetWare 5.1, Windows, Solaris, Linux
eDirectory 8.7	DS v10410.xx	NetWare 6, NetWare 5.1, Windows, Solaris, Linux, AIX
eDirectory 8.7.1		NetWare 6.5, NetWare 6, NetWare 5.1, Solaris, Linux, AIX, HP-UX

1.3 Requirements for Networks and the Internet

Networks and the Internet have many of the same requirements for a directory. As they both handle more and more users each year, they require simpler methods for handling users who need access to world-wide resources while maintaining security. eDirectory has been designed to fulfill these needs in the following types of directories:

- "Network Directory" on page 12
- "Cross Platform Directory" on page 12
- "Full Service Internet Directory" on page 13

1.3.1 Network Directory

Traditionally, networks have been organized around specific servers; if you wanted to access a particular service, you needed to access the server that held those services. This meant that you had to know the server's name and address, and you had to have a password stored on that server.

Instead of providing a server-centric environment, eDirectory provides a network-centric environment. The eDirectory database provides a view of all services, such as printers, servers, and volumes, available on the network. Once you log in to the network, you have access to all services you have rights to access. eDirectory provides the tools for administrators and applications to manipulate the network environment.

1.3.2 Cross Platform Directory

eDirectory is available on multiple operating systems: NetWare®, Windows 2000, Windows NT, Sun Solaris, Unix and Linux and IBM AIX. Usually networks that require the services of multiple operating systems also require the maintenance of a directory or a database of users and passwords for each operating system. eDirectory eliminates that kind of duplication and becomes the directory for the entire network.

1.3.3 Full Service Internet Directory

eDirectory is not just a network directory, it is also an Internet directory capable of handling all the requirements of a full service directory. A full service directory manages discovery, security, storage, and relationships.

Discovery. Discovery is the ability to browse, search, and retrieve specific information from the directory. For example, you can search for specific object types, such as users, printers, and application objects, or search for specific properties such as a user's name, phone number, address, or network number.

Security. Security controls access to all the information that is stored in the directory. You establish the rules and grant the rights that allow users to access the information in the directory. In addition, you control the flow of information within your company, across networks of partners, and even your customers. Using eDirectory, you can manage the electronic transactions between companies through the cryptographic and key management systems. More specifically, the Public Key Infrastructure (PKI) available in eDirectory today provides security for Internet data integrity and privacy across public networks. It includes both public-key cryptography and digital certificates for checking the authenticity of keys used in a public session.

Storage. Storage provides the basic ability to save information in the database for future reference. The database is indexed, cached, and guarded from data corruption by a transaction system. Besides merely storing data, the database allows you to automatically control the type of data by applying classifications to the data structures. The classifications are flexible and extensible to provide future representations in the database. The database can be split into physical pieces and distributed or placed on multiple servers. These features enable you keep a portion of the data close to the users and resources that need them and make multiple copies of the data for redundancy.

Relationship. Relationship is the ability to build associations between the people, devices, applications, and information on the network and the Internet. For example, instead of storing a user’s profile information on the local machine, it can be stored in eDirectory. The result is that the profile information becomes global within the scope of the directory and the user can access the profile information from anywhere. This means that the user receives the same profile regardless of where the user logs in to the network. In addition, the access to the profile information is tightly controlled. The only user that can gain access is the one with the proper credentials. Thus, the integrity of profile information is secure, and the user can access it globally, easily.

1.4 eDirectory Compliance to X.500 Standard

As directories have become global, companies have realized the need for directories which streamline network administration. Such directories need to be

- Replicated and distributed
- Compatible across companies, operating systems, and networks

Out of this need, the International Organization for Standardization (ISO) and the International Telecommunication Union (ITU) developed the X.500 standard.

X.500 specifies a naming and addressing structure, which functions at a global level because it is hierarchal and forms a logical tree. Each branch of the tree can represent a country or organization. eDirectory conforms to this structure as well as to the other requirements for the database, such as distinguished names, aliases, and the definitions for the types of entries and attributes that the database can contain. These definitions are called the directory schema, and X.500 allows for extensions to the schema definitions. eDirectory and eDirectory applications have added numerous object class and attribute definitions.

eDirectory differs from X.500 in that X.500 did not define any specific characters as separators for the components of a distinguished name, and eDirectory uses either a period or a slash for such a separator.

X.500 supports extensive query techniques that go far beyond standard name-to-address mapping functions. Since eDirectory was designed to be more than a name-to-address service, eDirectory supports these query techniques that return subsets of information about an entry. For example, eDirectory returns (1) the entry with all its attributes and values, (2) the entry with selected attributes and values, (3) the entry with all attributes and no values, and (4) the entry with selected attributes and no values.

Although in 1988 X.500 specified that the directory should be replicated and distributed, it did not specify a framework. In 1993, when X.500 specified protocols to use in synchronizing information between replicas, the first version of eDirectory was on the market. eDirectory uses the basic process of master/shadow, but it does not used the protocols defined in 1993. With the rise in popularity of LDAP (Lightweight Directory Access Protocol), standards bodies are concentrating on defining replication and synchronization extensions for LDAP. Novell is actively participating in these groups to help define these extensions so that they are compatible with eDirectory and other directories.

Access control either grants or denies access to a particular entry or attribute of an entry based on the requester's identity. A specification for access control did not appear until 1993 in X.500. Since the first version of eDirectory was already on the market at that time, Novell has continued to work with the standards bodies to define the framework for managing access control information.

1.5 eDirectory Compliance with LDAP v3

LDAP compliance involves a number of issues. LDAP is a protocol, but its specification also includes a directory schema and methods for extending its functionality through extensions and controls.

1.5.1 Protocol Compliance

The following table compares the protocol features of the newest release of the LDAP server in NDS 8.2*x* with two previous releases of the LDAP server in NDS 8 version 8.1*x* and NDS version 7.*xx*.

Table 1-2 *Protocol Compliance*

LDAP Feature	**eDirectory**	**NDS eDirectory**	**NDS 8**	**NDS**
	Version 8.6[x]	**Version 8.5*x***	**Version 8.1*x***	**Version 7.*xx***
Authentication (anonymous, clear text, SSL, and SASL bind)	Yes, for all but SASL	Yes for all but SASL	Yes for all but SASL	Yes for all but SASL

LDAP Feature	**eDirectory**	**NDS eDirectory**	**NDS 8**	**NDS**
Entry management (search, modify, compare, rename, adding, deleting)	Yes	Yes	Yes	Yes
Thread support	Yes	Yes	Yes	Yes
LDAP extensions	Yes	Yes	No	No
Readable root DSE	Yes	Yes	Yes	Yes
Referrals and traversals	Yes	Yes	Yes	Traversals—Yes Referrals—Single Configurable URL
Read and write schema	Yes	Yes	Yes for read schema	No
Auxiliary classes	Yes	Yes	Yes	No

Even though NDS version 7.*xx* is limited in its LDAP functionality, it complies with the LDAP v3 protocol specification because the LDAP server correctly responds to requests for unsupported features. NDS has increased its support of LDAP with each subsequent release.

For more information on LDAP and eDirectory see the NDK: LDAP and eDirectory Integration Integration Guide.

1.5.2 Schema Compliance

eDirectory and LDAP have had numerous differences in naming conventions, structural rules for containment, leaf versus containment classes, and supported syntaxes. Since NDS version 7.09, each release of NDS/eDirectory has extended its support of the LDAP schema.

For example, NDS and LDAP naming conventions for attribute and class definitions are quite different:

- NDS allows spaces in its schema definition names, usually capitalizes the initial letter of each word in the name, and supports a number of non-alphanumeric characters such as periods and colons.
- LDAP does not support spaces in a name and supports only one non-alphanumeric character for schema definition names, the dash (-). By convention, LDAP does not capitalize the initial letter of a name, but in multi-word names, it capitalizes the first letter of subsequent words.

Each release of the LDAP server has made these differences less significant. The first release of the LDAP server mapped the LDAP class and attribute names to their corresponding NDS class and attribute definitions. In NDS 8 version 8.1*x*, if the schema name is a valid LDAP name, mapping is no longer required. Missing attributes or classes have been added to the NDS schema using LDAP naming conventions, and classes have been modified to include the new LDAP attributes. In NDS 8, the schema supports auxiliary classes, and service packs make the auxiliary class feature compatible with earlier versions of NDS.

1.5.3 Extensions and Controls

eDirectory supports several LDAP extensions and controls, and handles unsupported controls according to the LDAP specifications. To determine supported extensions and controls query the server rootDSE.

1.6 Directory Access Protocols

NDS can be accessed through two directory protocols: NDAP (Novell Directory Access Protocol) and LDAP (Lightweight Directory Access Protocol).

1.6.1 LDAP

LDAP has been designed to access directory services. It is currently being developed and maintained by the IETF (Internet Engineering Task Force), an open, international community of network designers, operators, vendors, and researchers who define the standards and protocols that drive Internet computing. LDAP is not a transport protocol and has been implemented to use TCP/IP for a transport protocol.

NDS versions 5.73 through 6.*xx* support LDAP v2, and NDS versions 7.*xx* and 8.*xx* support LDAP v3 on NetWare, NT and Solaris platforms. In NDS 8 when LDAP clients formulate and send requests, an LDAP server listens for the requests, accepts them and sends them to NDS for processing. NDS processes the requests and returns the replies to the LDAP server which then sends them to the client. The LDAP server has become tightly integrated with the DS agent and is no longer an optional loadable module.

1.6.2 NDAP

NDAP is built on top of NCP (NetWare Core Protocol). Like NCP, NDAP is a request/reply protocol, but unlike NCP, it is a messaging protocol. Since an NDS request or reply can be larger than a single packet, NDS messages can be fragmented across multiple packets and reassembled by NDS before NDS hands the message to the NDS agent for processing.

NCP is not a transport protocol, and thus it relies on other protocols to transport packets on the wire. Since NDAP is built on top of NCP, any transport protocol supported by NCP is automatically supported by NDAP. In NetWare 4.x, NCP supports IPX™ and IPX encapsulated by IP (called NetWare/IP™) as its transport protocols. In NetWare 5.x, NCP supports IPX and IP as its transport protocol; the NetWare server can be configured to support IPX/SPX, TCP/IP, or both.

NDS 8 on NT and Solaris use TCP/IP as the transport protocol since TCP/IP is their native supported transport protocol.

SLP and SAP

In NetWare 4.x, NDS clients rely upon SAP to discover NDS trees and to advertise NDS tree names. In NetWare 5.x when a network is configured only for IP, NDS clients can rely on SLP (Service Location Protocol), if available, to discover NDS tree names, or the clients can be manually configured to know NDS tree names.

NDS uses SLP in two ways: advertise (or registration), and lookup (or name resolution). The NDS server (agent) advertises it’s IP referral addresses in an SLP record so others looking for a particular NDS tree by name may use SLP name resolution services to detect its presence. The NDS client (dclient) uses SLP name resolution to gain an entry point into the tree. That is, if it has no existing connections to the target tree, it will use SLP to locate a server in the tree (preferrably one that is close in heirarchical location to itself). After connecting to this server, it will use NDS resolve name functionality to lookup a server containing a replica of a partition containing that portion of the tree containing the target object. NDS resolve name protocol is always used first, if possible. Only in the condition that the client has no existing connections to any server in the tree will it go to SLP for that information.

DHCP

To make management of IP networks and IP addresses easier, NetWare 5.x includes a DHCP (Dynamic Host Configuration Protocol) server. The DHCP server provides dynamic IP address assignment so that many devices can share a limited address space. It also uses NDS to simplify the administration of IP address and device configuration information and passes this information to the hosts (DHCP clients) on the TCP/IP network.

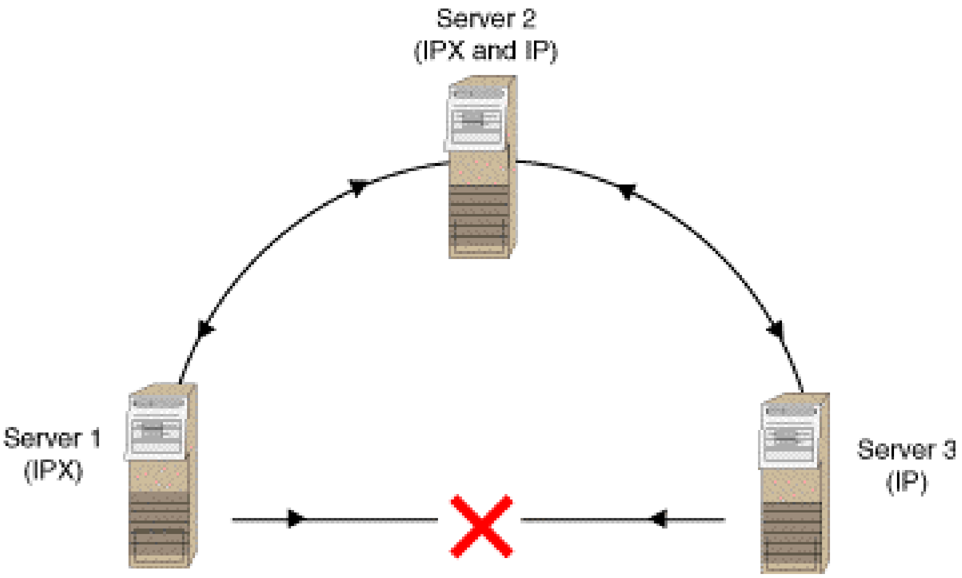
Mixed IP and IPX Networks

For NDS to work in a mixed IP and IPX network, the following modifications were made to NDS:

- Synchronization algorithms were changed to transitive synchronization
- Back links were changed to distributed reference links

These features eliminate the need for all servers in a replica ring to communicate directly with each other. The only requirement in a mixed IP and IPX network is that one server in the replica ring must support both IP and IPX.

Figure 1-1 *Server Synchronization Problems*



In the figure, Server 1 cannot synchronize directly with Server 3 because they are using different protocols. However, both Server 1 and Server 3 can synchronize with Server 2 and thus Server 2 becomes the intermediator. In this example, Server 1 will send its updates to Server 2. Server 2 will relay these updates to Server 3 and then update its transitive vector to indicate that Server 3 has received the updates from Server 1. The next time Server 1 and Server 2 synchronize, Server 1 will examine the transitive vector of Server 2 and then update its transitive vector to indicate that Server 3 has been synchronized.

Mixed NDS Version Networks

If an NDS replica ring contains servers running NDS versions 6.*xx*, 7.*xx,* and 8.*xx*, NDS automatically detects which synchronization process the server supports:

- If the server with the master replica is running NDS version 6.*xx*, the replica ring uses NDS 6.*xx* synchronization processes.
- If the server with the master replica is running NDS version 7.*xx* or 8.*xx*, the server uses transitive synchronization with the servers running NDS versions 7.*xx* or 8.*xx* and NDS 6.*xx* synchronization processes with servers running NDS version 6.*xx*.

1.7 Programming Interfaces for eDirectory

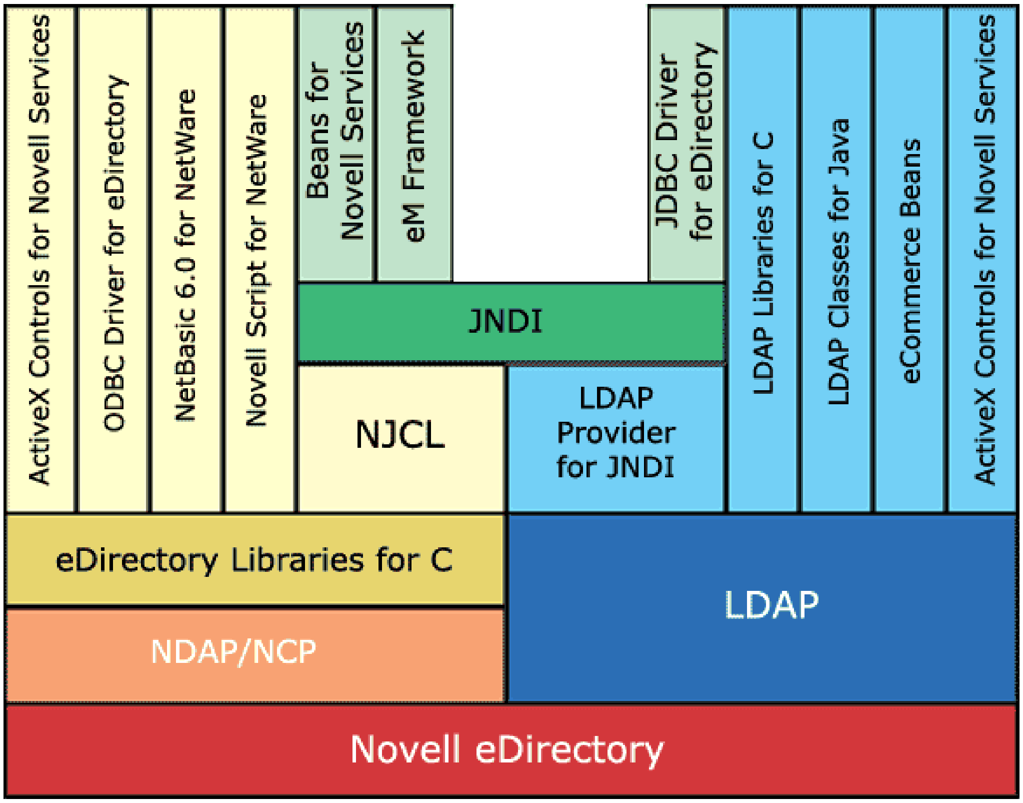
With the first release of NDS, application developers could access NDS through a set of C-interface functions. These functions were available on the server for NLM applications and on Novell clients for client applications.

Soon after NDS version 5.73 shipped, Novell made LDAP Services for NDS available. This product made the NDS database accessible by LDAP-compatible applications, such as browsers and Web servers.

1.7.1 NDAP and LDAP Interfaces to NDS

With NDS versions 7.*xx* and 8.*xx*, NDS became accessible through multiple interfaces. The following figure illustrates how NDS is accessible through two protocols (NDAP and LDAP) and which interfaces use which protocol.

Figure 1-2 *NDS Programming Interfaces*


ActiveX Controls for Novell Services
ODBC Driver for eDirectory
NetBasic 6.0 for NetWare
Novell Script for NetWare
Beans for Novell Services
eM Framework
JDBC Driver for eDirectory
LDAP Libraries for C
LDAP Classes for Java
eCommerce Beans
ActiveX Controls for Novell Services
JNDI
NJCL
LDAP Provider for JNDI
eDirectory Libraries for C
NDAP/NCP
LDAP
Novell eDirectory

These interfaces are available for NDS clients of NDS on NetWare, Solaris, and NT. Only NetWare supports a server-side interface for NLM development.

For developers that prefer scripting, NDS has the following interfaces:

- **JavaBeans.** This interface is a set of high-level abstractions of Java code that are packaged as functional, reusable components that can be run on any platform. The Beans for Novell Services and eCommerce Beans allow applications to use eDirectory for management, authentication, and access control.
- **ActiveX.** This interface is used by many visual programming tools such as Visual Basic and Delphi. The Novell Controls for ActiveX allow you to log in and out of the NDS tree, to read, write, and modify information in the NDS tree, and to access the NDS schema so that you can create new attributes for existing objects and for new objects. Two versions of the directory control are available: one through NDAP and one through LDAP.
- **ODBC.** The ODBC Driver for eDirectory is a driver that enables SQL (Structured Query Language) to access NDS using visual programming tools such as Visual Basic and Delphi. The ODBC Driver for eDirectory allows you to use standard database utilities for reporting, graphing, and charting NDS information. It is a read-only driver.

- **NetBasic.** This interface allows applications to access NDS through a Visual Basic script-compatible language called NetBasic. Novell NetBasic 6 allows you to build custom scripts to handle NDS administrative tasks.
- **Novell Script.** Novell Script for NetWare (NSN) incorporates predefined NetWare components with a VBScript syntax-compatible scripting language. NSN provides a unified scripting and component architecture. The UCS (Unified Component System) kernel allows you to write solutions using and combining NSN components, JavaBeans, Perl scripts, and other popular scripting and component languages.
- **JDBC.** The Novell JDBC Driver for NDS is a Java Database Connectivity driver that enables Java programs to execute SQL (Structured Query Language) statements to access NDS. JDBC is similar to ODBC, but it has been designed to work specifically with Java programs and used LDAP to access eDirectory.

For developers who prefer object oriented programming, the following interfaces are available:

- **LDAP Classes for Java.** This interface allows access to NDS over an LDAP client rather than a Novell client. It supports all the standard LDAP functions for LDAP v3 as well as LDAP extensions for managing naming contexts (NDS replicas and partitions). For secure authentication, it currently uses SSL (Secure Socket Layer).
- **JNDI.** The Java Naming and Directory Interface is an addition to JavaSoft's Enterprise API set and provides an easier way within Java to discover network resources. JNDI allows you to access NDS objects, partitions, and schema using LDAP.
- **ConsoleOne.** Novell ConsoleOne is a Java-based framework or shell with APIs for creating network management utilities, or snap-ins. All aspects of NDS are accessible, making it possible for you to access NDS easily and efficiently as you create a management snap-in for your NDS application.

For C programmers, the following interfaces are available:

- **NDS Libraries for C.** This interface allows access to all the features of NDS: read, write, and modify NDS object and attribute information; schema extensions; and partition and replica management. The functions are available for the development of both NDS client dlls and NDS server applications.
- **LDAP Libraries for C.** This interface allows access to NDS over an LDAP client rather than a Novell client. It supports all the standard LDAP functions for LDAP v3 as well as LDAP extensions for managing naming contexts (NDS replicas and partitions). For secure authentication, it currently uses SSL (Secure Socket Layer) or SASL (Secure Authentication Security Layer).

For additional information on, and access to, these interfaces, see the Novell Developer Kit (http://developer.novell.com).

1.7.2 Compatibility Criteria for NDAP Applications

For NDAP applications to pass the compatibility criteria for NDS, the applications must meet the following standards.

- **Installation.** You must be able to install the application in any container. The application's install utility must only require ADMIN rights. It must not require the user to log in is as ADMIN.

- **Bindery.** The application must not use bindery calls. NDS applications must use DS naming conventions exclusively.
- **Searches.** The application must not perform unlimited scope searches of the entire NDS tree; instead it should limit searches to segments of the tree. Searching the entire tree causes unacceptable degradation of NDS performance
- **Compare versus Read.** Whenever possible, the application must use a compare API in place of a read API. This ensures that you are not forcing a reference to the main NDS tree and adversely affecting the performance of NDS. A local reference to a local replica is much faster.
- **Polling Objects.** The application must not poll NDS objects. A registered event must be used to signal an attribute change. Polling has a negative effect on performance.
- **Schema Extensions and Objects.** You must registered with Novell (http://developer.novell.com/support/) to obtain a prefix and ASN.1 ID for your extensions.The names must use the prefix and conform to LDAP naming conventions.The NDS object class definitions and directory objects created by your application must be referenced and used by your application.
- **Duplicate Information.** The application must not duplicate information already stored in NDS (for example, create and maintain its own list of user objects). It is better to leverage the data that already exists, which will improve the performance of your application and NDS.
- **Authentication.** The application must authenticate exclusively through NDS. The application must not save NDS passwords or leave passwords in memory for any length of time.
- **Security.** The application should not create any back doors that enable access to the applications' supervisory rights without logging in. The application's objects should not pose any security threat to the system.
- **Objects at the Root.** The application must not create or use an object at the root of the tree that might cause security issues, network administration difficulties, or inconvenience to users that have insufficient rights to the root of the NDS tree This must be accomplished without reducing the security level of the NDS root or employing object-specific security rights at the root of the tree.

1.7.3 Compatibility Criteria for LDAP Applications

For LDAP applications to pass the compatibility criteria for NDS, the applications must meet the following standards.

- **Installation.** You must be able to install the application in any container. The application's install utility must only require ADMIN rights. It must not require the user to log in is as ADMIN.
- **Schema Extensions and Objects.** You must registered with Novell (http://developer.novell.com/support/) to obtain a prefix and ASN.1 ID for your extensions.The names must use the prefix and conform to LDAP naming conventions.The schema definitions and entries created by your application must be referenced and used by your application.
- **Mapping.** If your application supports releases of NDS previous to NDS eDirectory (8.2x), your application's installation program must either map the schema extensions to NDS classes and attributes or you must provide detailed documentation that instructs the system administrator how to perform the mappings manually.
- **LDAP Extensions and Controls.** Your application must use only LDAP extensions and controls that are compliant with LDAP v3 specifications.

- **Polling Objects.** Your application should not poll NDS objects for changes. Polling has a negative effect on NDS performance.If you cannot eliminate the need for polling, do it infrequently and allow the system administrator to set the polling interval.
- **Duplicate Information.** Your application must not duplicate information already stored in NDS (for example, create and maintain its own list of user objects). It is better to leverage the data that already exists, which will improve the performance of your application and NDS.
- **Internationalization.** Your application must use UTF8 for message tables, which provide compatibility with language support for users world-wide. This support is required even if your messages are in English.
- **Security.** Your application should not create any back doors that enable access to the applications' supervisory rights without logging in. The application's objects should not pose any security threat to the system.
- **Objects at the Root.** Your application must not create or use an object at the root of the tree that might cause security issues, network administration difficulties, or inconvenience to users that have insufficient rights to the root of the NDS tree This must be accomplished without reducing the security level of the NDS root or employing object-specific security rights at the root of the tree.
- **Authentication.** Your application must use secure connections. It cannot authenticate using clear text passwords. The application must not save NDS passwords or leave passwords in memory for any length of time.

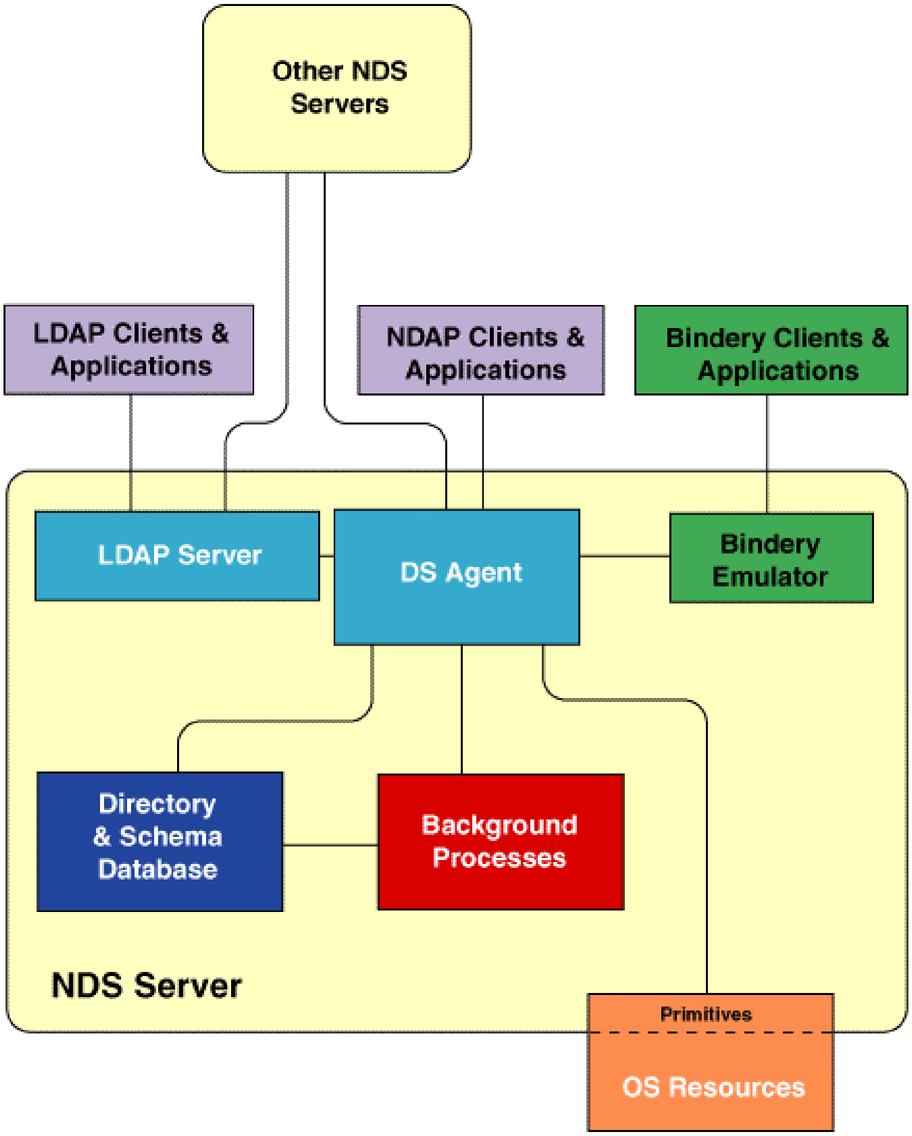
For more information about NDS Test Tools, see Novell Software Test Tools (http://developer.novell.com/ndk/softtestv3.htm)

1.8 NDS Architecture

The DS agent within an NDS server processes requests from three types of clients. Two, the NDAP and LDAP clients, have similar functionality and have full access to the directory, its entries, schema, operations, and background processes. The third client, the bindery client, has restricted access and must go through a bindery emulator which makes the directory appear as a flat bindery database and hides all functionality that isn't available in the NetWare 3.x bindery. See "Bindery Services" on page 93 for more information about bindery emulation.

The DS agent also communicates with other NDS servers. The agent establishes a client connection with another NDS server and uses the connection to read, write, and search entry information, to perform partition operations, and to synchronize data. The following figure illustrates these communication paths.

Figure 1-3 *NDS Communcation Paths*


Other NDS Servers
LDAP Clients & Applications
NDAP Clients & Applications
Bindery Clients & Applications
LDAP Server
DS Agent
Bindery Emulator
Directory & Schema Database
Background Processes
NDS Server
Primitives
OS Resources

1.8.1 LDAP Clients and Applications

LDAP clients and applications currently interface with NDS through the Novell LDAP server. The LDAP server communicates directly with the DS agent on its server as well as with DS agents on other NDS servers. The client determines whether NDS returns referrals or uses referrals to traverse the tree and go remote to find the information on other NDS servers.

Since LDAP clients and applications do not require Novell client software, the LDAP application is responsible for establishing a connection and authenticating to the NDS server. It is also responsible for ensuring platform dependencies are met. For example, an LDAP application that uses Java and runs on a client workstation must have a JVM installed. An LDAP JNDI application must have a Java service provider installed.

1.8.2 NDAP Clients and Applications

NDAP clients and applications require Novell client software which includes support for various languages (C/C++, Java, and JNDI) and includes a JVM and a Java service provider. Scripting components have been built on top of these languages to allow additional methods for NDS access.

The client software establishes and manages the authentication to the NDS tree. It formulates the application's request into an NDAP request that is sent to the DS agent. The application can use the connections established by the client software or it can establish its own connections.

1.8.3 DS Agents

DS agents are responsible for managing the information stored in the NDS database and coordinating distributed operations with other servers. The agents manage all NDS requests, including the following:

- Security (authentication and access control)
- Entry management (add, delete, modify, search, read)
- Partition operations (split, join, move)
- Replica operations (add, delete, change type)
- Replica and schema synchronization
- Schema management (read and write)

1.8.4 Directory and Schema Database

The NDS database contains two main types of information: directory and schema. The directory contains entries with their attributes and values. Novell applications usually refer to entries as objects and attributes as properties. For more information on how NDS organizes, uses, and allows access to this information, see "eDirectory Objects" on page 25.

The schema portion of the database contains the object class definitions and the attribute definitions. These definitions control the information that can be added to the directory. For example, the schema contains a definition for a User object. This definition determines where in the NDS tree a user entry can be located, what the user entry can be named, and what attributes must have values before a user entry can be created.

1.8.5 Background Processes

The DS agent communicates with the background processes that keep the NDS database synchronized with other NDS servers and purged of obsolete data. These processes run without user intervention, although some allow for limited user management. For more information about the processes, see Section 4.7, "Background Processes," on page 88. For information on commands for managing the processes, see "Directory Services Trace Utilities" on page 125.

1.8.6 OS Resources

The NDS server accesses operating system resources, such as disk space and memory, through a primitive layer interface. This layer isolates this functionality so that NDS can be implemented on multiple operating systems such as NetWare, NT, and Solaris.

eDirectory Objects

2

eDirectory represents network resources as objects in the eDirectory tree. For example, objects can represent organizations, people, or physical resources such as printers and workstations.

The eDirectory schema dictates how these objects can be named, placed in the tree, and manipulated.

2.1 eDirectory Names

eDirectory names are made up of a series of components and their respective types. A typical eDirectory name would look something like CN=JRoss.OU=Engineering.O=ACME. The components of this name represent objects in the tree and their types represent the naming attribute of that object. So in this example, JRoss is the name of a user object and CN is the naming attribute for a user object. Engineering is the name of an Organizational Unit and OU is the naming attribute for an Organizational Unit. Each component in the name is separated by a period.

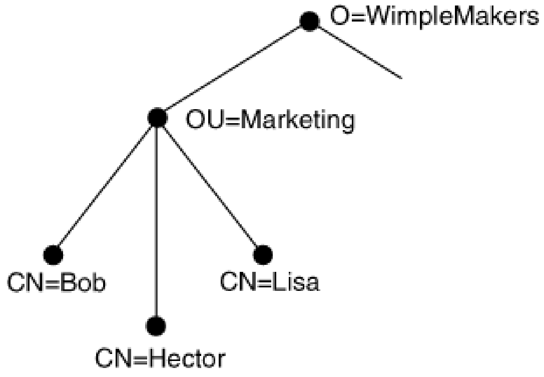
See Also:

- "Hierarchical Naming" on page 25
- "Alias Naming" on page 27
- "eDirectory Context" on page 27
- "Character Encoding" on page 28

2.1.1 Hierarchical Naming

The arrangement of names in the eDirectory tree reflects the hierarchical relationships that exist among objects. The following figure shows an example of an eDirectory tree. All the objects of this particular tree are subordinate to the Organization (O) WimpleMakers, which is referred to as the partition root.

Figure 2-1 *Example eDirectory Tree*



WimpleMakers is actually subordinate to an object called [root] All eDirectory trees have [root] as the topmost object.

The Organization Unit (OU) Marketing is subordinate to WimpleMakers. At the bottom of the tree are the names of three employees.

As you study this tree, you can observe several things. First, if you begin with any object in the tree and follow the names back to the root, you trace a unique path of names. For example, the naming path for Bob would include Bob, Marketing, and WimpleMakers. The complete naming path from a particular object to the root of eDirectory is called the object's distinguished name (DN), or complete name. The distinguished name forms a unique reference that identifies the location and identity of an object in eDirectory.

Another feature of the tree is that each name is identified by type. The name WimpleMakers is of type Organization, Marketing is of type Organization Unit, and so on. The Schema specifies the rules for object relativity within the tree. For example, an object of type Common Name can be subordinate to an Organizational Unit, but not the other way around.

The individual name assigned to an object is called the object's relative distinguished name (RDN), or partial name. The partial name must be unique in relation to the object's superior. In our example, there can be only one object named Marketing that is subordinate to WimpleMakers, one object named Bob that is subordinate to Marketing, and so on.

A relative distinguished name can include the name's type specification. The type and its value are joined by an equal sign:

```
Common Name=Bob
```

Using an abbreviated attribute type, the above name is:

```
CN=Bob
```

If a partial name includes more than one naming attribute, the names are separated by a plus sign:

```
L=NPD+S=Utah
```

When expressing part or all of a complete name, the partial names are separated by periods. A complete name requires type specifications for each partial name that can be specified explicitly, as shown below:

```
CN=Bob.OU=Marketing.O=WimpleMakers
```

Names without the types explicitly shown are called typeless names as shown below:

```
Bob.Marketing.WimpleMakers
```

The types are implied based on the default context:

Components of a complete name that are further from the root are referred to as less significant, while those closer to the root are more significant. Thus, Bob is the least significant component in the complete name above, while WimpleMakers is the most significant.

eDirectory reserves a few character symbols, for denotative purposes. The reserved characters are the period (.), the comma (,), the equal sign (=), the plus sign (+), and the backslash (\).

If these characters are a part of a name, then they must be escaped with a backslash (\). To show the period in "Inc." you would use the backslash, as shown in the following example:

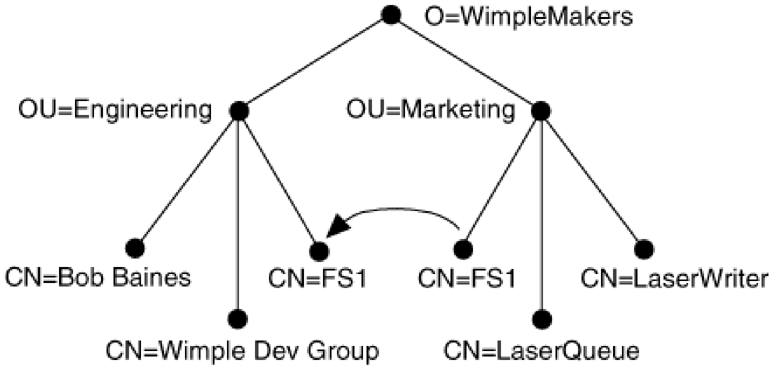
```
OU=Bearskins,Busbies,and Green Berets,Inc\..
    O=WimpleMakers
```

2.1.2 Alias Naming

An alias is a name that can be used as a substitute name for identifying an object. The alias provides an alternate naming path for locating the object in eDirectory. Any object in eDirectory can be aliased. If an object has subordinates, its alias appears to have the same subordinates. However, the location of the alias itself must be a leaf node (a node having no subordinates) in the eDirectory tree.

The following figure illustrates how aliases can be used. In this example, the server FS1 is found in the subtree Engineering. However, if users in the Marketing subtree access this server often, they may find it convenient to create an alias for the server under Marketing. That way, the server FS1 can be referenced relative to a local name context ("OU=Marketing.O=WimpleMakers") that is already set rather than establishing a new context ("OU=Engineering.O=WimpleMakers").

Figure 2-2 *Using Alias Names*



Aliasing is a convenience that has many applications. For example, aliases can be used to simplify searches of eDirectory. An administrator could create aliases in a particular subtree for all the modems on the network so users would have to search only that one subtree to receive information about all available modems.

2.1.3 eDirectory Context

An eDirectory context is a reference point into an eDirectory tree. The complete name of the container pointed to is called the name context, and serves as a default naming path for eDirectory operations. When a workstation's requester is loaded, the name that is used for the context comes from the client networking configuration files.

An object's context can be expressed either as a Distinguished Name (DN) or as a Relative Distinguished Name (RDN). The DN is similar to a file name with the full path. The RDN is the partial name relative to the current context. For example: a user JSmith exists in the container HR.Novell. If you set the current context to HR.Novell, the RDN is JSmith and the DN is JSmith.HR.Novell. However, if you set the current context to Novell, the RDN is JSmith.HR and the DN is still JSmith.HR.Novell. In fact, the DN is always the same regardless of the current context.

Understanding how DNs and RDNs are used is important, especially if you are writing an application that is going to "walk the tree." If you save the RDN and then change the current context, that RDN becomes invalid and an error is returned. However, if an object with the same CN exists in the new context, you don't get an error, but you could get the wrong data. For example, you store the name JSmith and your current context is HR.Novell. You then change the current context to Security.Novell and there happens to be a JSmith whose DN is JSmith.Security.Novell. You would get the results of JSmith.Security.Novell and not JSmith.HR.Novell. No error would be returned, but you would receive the wrong data.

The name context is implemented as a value in the operation's eDirectory context variable that is created with NWDSCreateContextHandle. Initially, the value of the name context is taken from a global name context created as follows:

- If the application is running on a workstation, the requester's configuration file is used to initialize the global name context.
- If the application is an NLM™ application, the global name comes from the server's Bindery context setting that is set with the set bindery context = command. This setting can be entered at the system console, or it can be placed in the server's STARTUP.NCF file.

When an application creates an eDirectory context, the name context field takes its initial value from the global name context. Thereafter, the name context for the current session can be modified using the eDirectory functions. The eDirectory functions do not modify the requester's context setting or the server's bindery context.

Any partial names submitted by a client for a particular request can be expressed relative to the requester's name context. The client agent examines each name and expands it accordingly before sending the name to eDirectory.

The following example shows how the name context is typically applied to input name:

If the name context is

```
OU=Marketing.O=WimpleMakers
```

and the client submits the partial name

```
CN=Bob
```

the client agent appends the name context to the partial name, thus forming the complete name

```
CN=Bob.OU=Marketing.O=WimpleMakers
```

2.1.4 Character Encoding

Character sets familiar to humans are represented in computers by a sequence of bit settings. A complete set of computer representations mapped to human readable characters is stored in memory in what is called a code page.

Unfortunately, the same set of characters may have different computer representations. One computer might send a set of bits with binary value 65 representing the letter "A" but the receiving computer's code page interprets 65 as something entirely different.

To resolve this and other multilingual translation problems, a group of computer companies formed the Unicode* Consortium. They established a 16-bit representation for almost every character used in any language. Subsequently, the ISO adopted the Unicode set as a subset of its 10646 specification, which is meant to do the same thing.

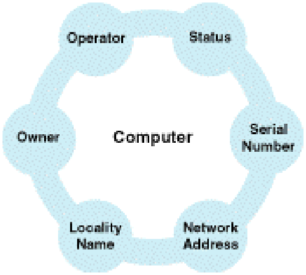
All eDirectory information is stored and transmitted in Unicode representation. Since not all computers today know how to convert Unicode into human readable characters, a translation from Unicode to the computer's local code page representation is often needed. By default, the library converts all strings from Unicode to the local code page representation, but this conversion can be disabled if desired.

2.2 Types of Information Stored in eDirectory

The information stored in eDirectory can be read as well as modified by users who have been granted the applicable access rights.

The information stored in eDirectory is a collection of objects with associated attributes (also called properties) which can have values assigned them. The following figure shows an object called Computer with some associated attributes (such as Operator, Status).

Figure 2-3 *An Object as a Set of Attributes*



The objects in eDirectory and their names correspond to things that humans relate to when dealing with computers, networks and information. Objects such as countries, organizations, people, computers, and so on are types of objects you might find in eDirectory. Specific objects might be such objects as US, Novell, Fred, FS1, or Print Server. The objects are intended to fit into a human being's conception of things, not necessarily a computer's.

Informally, much of the attribute information found in eDirectory falls into one of four categories: Names, Addresses, Membership Lists, and Descriptions. Although these categories are not formally denoted in eDirectory, they are helpful in understanding typical kinds of information that users and developers can find in eDirectory.

Lists Stored in eDirectory. eDirectory can be used to store and manage lists of objects. A list is associated directly with a specific object. Typically these lists contain names of other objects in eDirectory. These lists might be used for distribution or authorization purposes. For example, a distribution list can be used to determine who should receive electronic mail messages or who is qualified to perform a specific function. A list name identifies it as a property of an object. The list itself is the value of the property.

If desired, members in the list can be granted the privilege of adding or deleting themselves from the list. This privilege can be used to make a list self-moderating. Members of the list can decide for themselves whether or not they want to be on the list and receive the related information.

For authorization purposes, a membership list might be used to define the objects that have access to a resource. For example, a server can have a list of operators who are authorized to maintain it. Or, a queue object can have a list of users authorized to place entries in the queue. eDirectory provides several standard lists for specific objects, such as users and servers. Applications can also define lists for their own special needs.

Descriptions Stored in eDirectory. eDirectory can be used to store object descriptions. A *Description* property can be used to associate a descriptive string with an eDirectory object. The string can contain any information that describes the object. There are various properties that provide specific kinds of descriptive information, such as the serial number of a device or the fonts supported by a printer.

Addresses Stored in eDirectory. Like a published directory, eDirectory provides additional identifying or qualifying information about an object. Some examples are postal addresses, electronic mail addresses, telephone numbers, physical locations, and network addresses.

You might need to change the location information from time to time. Such changes to an object are synchronized throughout eDirectory. By specifying an object by name, a user can reliably locate the object despite alterations in the object's physical location or changes to the physical layout of the network.

Names Stored in eDirectory. The most significant piece of information associated with an object is its name, which identifies the object within the context of eDirectory. Names are intended for humans; that is, they are character strings that humans can read and remember. The name of an object is a property of the object.

Names are used as keys to the information stored in eDirectory. A person using eDirectory can supply the name of an object and receive information that describes the object. Or, a user can supply a description and receive a list of names of objects that fit that description. By employing natural naming conventions, eDirectory makes it easy for people using eDirectory to obtain less intuitive kinds of information.

2.3 Retrieval of Information from eDirectory

The details for accessing eDirectory are implemented at the application level. eDirectory provides a complete set of functions to support the development of directory applications. However, eDirectory anticipates that users will take several approaches to obtain information from eDirectory. These approaches include lookup, browsing, and searching.

Lookup in eDirectory. Lookup is probably the most obvious approach to searching eDirectory. The user identifies a particular object and eDirectory is called on to find it. The object is identified by its name or by a convenient alias. Along with the name, the user will usually specify one or more attributes. If the object is found, eDirectory can return the current values of the specified attributes. To receive the information, however, the user must possess sufficient access privileges to the object. Lookup is supported by the Read and Compare services detailed in the eDirectory API.

Browsing eDirectory. At times a user might want to browse areas of eDirectory. Perhaps the user does not know the precise name for an object but has an idea of where the object is located in eDirectory. Or, a user might simply be interested in learning what objects are found in a given part of eDirectory. Browsing, like lookup, requires that the user has sufficient privileges with respect to the area of eDirectory the user is investigating. Browsing is supported by the NWDSList and NWDSSearch functions.

Searching in eDirectory. Searching is one of the more powerful tools provided by eDirectory. This allows the user to search for specific categories of information. The user specifies the criteria for the category, and eDirectory returns information about objects that satisfy that criteria. The user also specifies the scope of the search and the type of information to be returned. For example, a user could specify a search for the names of all the PostScript printers in the Advertising Department. Searching is supported by the *Search* service detailed in the eDirectory API.

Use the backslash escape character to when searching for names containing either the asterisk (*) or the backslash itself (\\).

2.4 Tree Walking

When a client agent submits a request to eDirectory, the request is not always received by a name server that is qualified to fulfill the request. The name server receiving the request must find a name server that can fulfill the request. Several name servers may need to be contacted before a qualified server is located.

To find the information, a name server initiates a search until a replica is found that contains the desired information. This process is called tree walking.

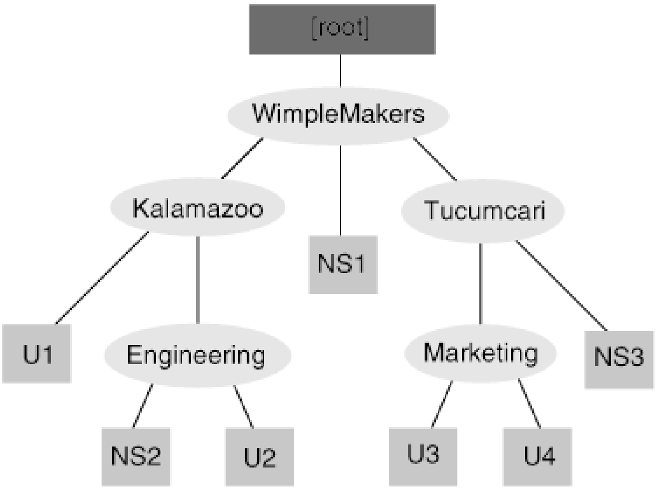
All eDirectory requests can be broken down to one or more names identifying objects for which information is needed. In pursuing each name component in a request, the name server searches for a partition that contains some part of the name path associated with the request. Once a partition is found, the search moves from that partition to the partition that actually contains the item. Until a relevant partition is found, the search proceeds toward the eDirectory directory [root], since any request can be pursued successfully by beginning at the [root].

If a name server can find no other information, it can at least provide a name server that has a partition with information about objects that are closer to the [root] than itself.

The tree walking process relies on subordinate references to connect the tree. The subordinate reference is an entry in a superior partition that points to a subordinate partition. There is a subordinate reference for each partition that is subordinate to a given partition. When walking the tree, a name server is given the name of an object. From there, the name server decides which server to attempt to access next in order to locate the object.

The following illustration shows an example eDirectory tree. It shows the total tree structure with no partition designations, and does not give any indication as to which servers the partitions and/or replicas are stored on.

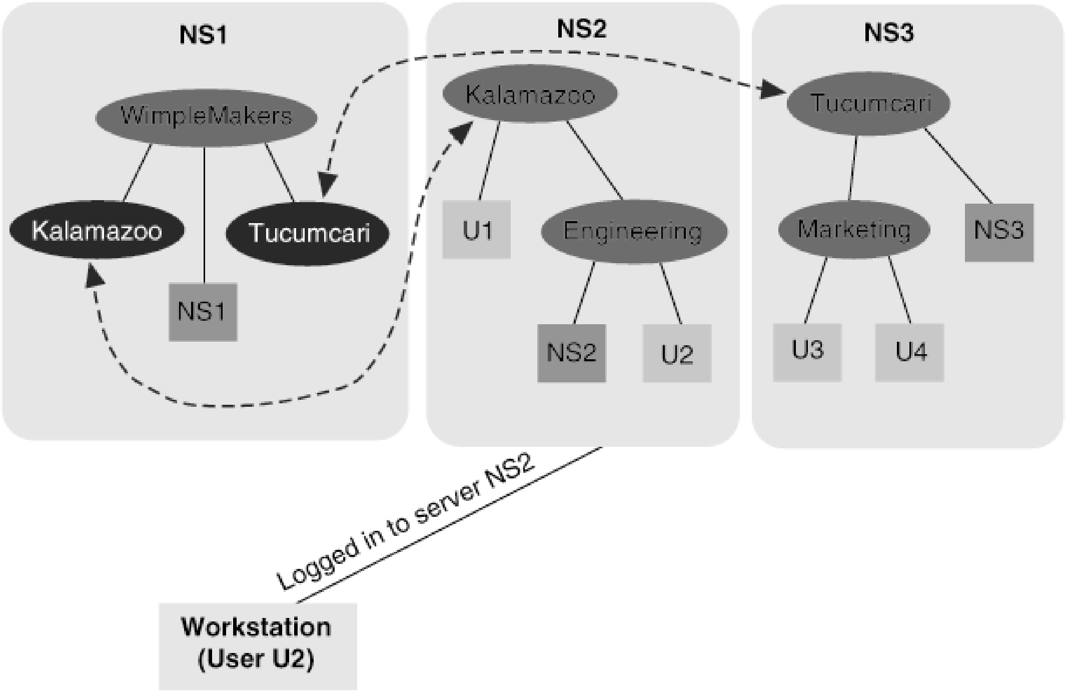
Figure 2-4 *Sample Tree Structure*



This tree has three partitions. WimpleMakers, Kalamazoo, and Tucumcari were the objects selected as the partition roots, and NS1, NS2, and NS3, respectively, were chosen to store the partitions.

Suppose a client (user U2) has a connection with the name server NS2 and makes a request to update the Marketing object. It specifies the name Marketing.Tucumcari.Wimplemakers in the request. NS2 has no information about Tucumcari or Marketing, so it must begin a tree walking process. NS2 passes the request up the tree to NS1, because it knows it is closer to the [root] because of its superior reference. This short walk up the tree is the only upward walk that must occur for this particular request, as the following figure shows.

Figure 2-5 *A Walk Up the Tree*



After locating Tucumcari, which is part of Marketing's distinguished name, eDirectory begins the walk down the tree toward the desired object. NS3 then completes the tree walk when it successfully identifies Marketing as part of its partition. In this arrangement, each of these partitions is a master partition and can be written to, so NS3 is able to fulfill any request allowed by the eDirectory security access control lists.

If other servers were added to this tree, they could be designated as replica holders of any one or all of the partitions. If more replicas existed, server NS1 would have to determine which server to send the request to.

Progress Reports. During the tree-walking process, when a server receives a request and discovers information about how to get closer to the desired partition and server, this information is sent back to the server that sent the request. This progress report is called a referral.

The referral gives the requesting server a new list of name servers to try if the name server cannot satisfy a request.

A referral contains the following fields:

Server List

This field is a list of name servers. The servers initially set this field based on currently connected servers and SAP.

Class Definition Cache

This cache holds the expanded class definition of each eDirectory class.

2.5 eDirectory Object Management

No business or organization stays the same over time, so eDirectory allows users to modify their Directory database by adding, deleting, and modifying entries. When a client adds or changes entries in the tree, eDirectory ensures that the entries conform to the standards set in the schema, thus making sure entries are consistent across the network.

Before an operation can begin, a client must obtain the Entry ID of an entry held on the server from which it is requesting the operation. Each Entry ID is valid on a specific server. For example, two servers, A and B, could hold replicas of the same entry, CN=John. The Entry ID for the entry CN=John could be different on server A than it would be on server B. An Entry ID is valid indefinitely on a specific server.

Adding an Entry. This operation allows you to add an entry or an alias to eDirectory. The schema dictates where entries are created and what their attributes can be. To be valid in the Directory, the new entry must hold the attribute Object Class, which has as its value a valid eDirectory base class. The object classes are defined in the schema. These object classes specify:

- Mandatory attributes for an entry
- The attributes that can appear in its Relative Distinguished Name
- Optional attributes, if any

The mandatory attribute fields for the new entry must contain values, or the operation fails. For example, a User entry must have values for the CN, Surname, and Object Class attributes to be a valid entry. All of these attribute values must conform to the syntax associated with the attribute.

The new entry's parent entry must a valid container. Also, the parent's base class must be in the containment list of the new object's base class. This parent entry must be a valid container, or the operation fails.

Comparing Attribute Values. The Compare operation compares a given value to an existing attribute and its value. For example, the client might use Compare to find out if the group Everyone.Novell has a Member attribute that has a value of Admin.Novell. The operation returns true if the values match or false if they do not. This operation verifies the existence of a value without having to read the value.

Comparisons are based on the matching rules of the attribute syntax assigned to the attribute. Therefore, the matching rules must provide for this operation.

Modifying an Entry. The Modify Entry operation allows a client to modify, add, or delete an entry's attributes in the same operation, except naming attributes. For example, the client might use Modify Entry to modify a user's telephone number.

All modifications must obey the rules defined by the class of the entry. For example, read-only attributes and “hidden” attributes cannot be changed. Because the entry's object class is read only, the class cannot be changed.This operation can also be used to modify an alias.

Modifying an Entry's Name. The Modify RDN operation allows a client to change an entry's Relative Distinguished Name (RDN). Using this operation, a client can specify a new naming attribute and value that will serve as the entry's RDN, or the operation can specify a new value for the existing naming attribute. In either case, the new name must obey the rules for the object classes of the entry being modified. Only a few object classes currently define more than one naming attribute.

If the operation changes a multivalued attribute or substitutes a new naming attribute, the old value can remain in the entry if the client requests it. In that case, the former attribute value will not remain part of the entry's RDN, but will remain as an attribute value or values of the entry.

Moving an Entry. The Move Entry operation comprises two routines, Begin Move Entry and Finish Move Entry, that place an entry in a different location in the Directory tree, which changes the entry's Distinguished Name.

Reading an Entry. The Read operation returns information about an entry stored in the Directory. It reads attribute information associated with the entry. The object class determines which attributes the entry has.

The client can choose to return information about:

- Specific attributes or a complete list of an entry's attributes.
- Attribute names only or attribute names and values.
- Effective access control privileges of the current user or of another user.

Removing an Entry. The Remove Entry operation removes an entry from the Directory tree. The entry being removed must be a leaf or a container entry having no subordinates. If the entry has subordinates, the operation fails.

A client or server can use Remove Entry to remove an alias from the Directory tree. I f the entry ID indicates that the entry is an alias, the operation removes the alias, not the entry the alias refers to.

eDirectory Security

3

Novell® eDirectory™ security is extremely powerful and flexible. A network administrator can control access from any object in the eDirectory tree to any other object in the tree. Any two users—groups, containers, or any other eDirectory object—might be able to access the same network resources in the tree in completely different ways based on their rights. Or, they might not even be able to “see” the same objects at all.

The following sections describe how eDirectory provides security:

- Section 3.1, “Authentication,” on page 35
- Section 3.2, “Access Control Lists,” on page 42
- Section 3.4, “NetWare File System,” on page 52

3.1 Authentication

Authentication is the process by which users establish their identities when accessing a network application service. Authentication

- Prevents unauthorized objects from gaining access to the eDirectory tree
- Ensures that authorized objects gain proper access.

When an eDirectory tree is first created, a user named Admin is created in the Root of the tree. This user has all rights to all objects in the tree. This ensures that the eDirectory tree contains a user with the required rights to create and modify objects and administer the tree. Four other entities are also created, [Public], [Root], [Self], and [Creator], which are used to grant access to eDirectory information.

Before users can access any information, the users must attach to an eDirectory server. This initial attachment, sometimes called a connection, allows them to access the information that has been made accessible to [Public]. By default, this information is what is required for a user to log in or authenticate. However, the system administrator can grant additional rights to [Public], rights which are automatically given to any user who connects to an eDirectory server in the tree.

Attaching to a server establishes a connection. When users *log in,* they establish their credentials. Their connections must be *authenticated* in order for them to access all of the eDirectory information for which they have rights. All authenticated users have rights to what has been made accessible to [Root] and [Self].

Authentication has three phases:

- Login—Gets the private key.
- Client-server authentication—Uses the signature to generate a proof that is used to establish the client’s identity and authorizes it to access network services.
- Background authentication—Uses the signature and proof to access additional network services.

Authentication is transparent to the user. During login, the user enters a password, and the remainder of the operation is performed in the background by the authentication functions.

Authentication is session-oriented. The data that provides the basis of authentication is valid only for the duration of the current login session. The critical data used to create authenticated messages for a particular user is never transmitted across the network.

eDirectory uses two types of authentication methods, depending upon the protocol used to establish the connection:

- NDAP (Novell Directory Access Protocol)—a protocol built on top of NCP. Novell client software uses NDAP to authenticate to the eDirectory tree and additional NCPs to license the connection and access the file system. For more information, see "NDAP Authentication" on page 36.
- LDAP (Lightweight Directory Access Protocol)—a protocol built for directory access and used by most internet browsers. For more information, see "LDAP Authentication" on page 42.

3.1.1 NDAP Authentication

Authentication relies on encryption systems, procedures that allow information to be transmitted in unreadable forms; that is, it is not meaningful to anyone unless they have the key to decrypt it. Authentication uses encryption to produce the information that can authenticate a client to a service. Encrypted information is used during the initialization phase of authentication, as well as in the authentication itself.

Typically, an encryption process uses two inputs: the data to be encrypted and an encryption key. The result is an unreadable message called the ciphertext. Additional input, such as a session identification, can also be added to the encryption process to provide an additional context for the message. The session value ensures that the message is part of a current, ongoing dialogue and has not been counterfeited from another session.

Deciphering an encrypted message by reversing the encryption process is called decryption. The ciphertext and the decryption key are fed into the decryption process, and the result is the original message. The message can be reassembled from the ciphertext only by using the correct decryption key.

The following sections describe additional details of the authentication process:

- "Public Key Encryption" on page 36
- "Authentication Requirements" on page 37
- "Authentication Process" on page 38
- "Background Authentication" on page 39
- "Password Changing in eDirectory" on page 40
- "Changing Public and Private Key Pairs" on page 41

Public Key Encryption

Public key encryption uses two keys, one to encrypt and the other to decrypt. The relationship between the two keys is mathematically complex, so it is virtually impossible to infer the value of one key from the other. Consequently, one key can be made public without the risk of exposing the other key.

The network entity that will receive messages generates a key pair and distributes the public key. Services that send encrypted messages must do so with the public key, and the holder of the private key can then decrypt the messages.

If you have the public key, you can encrypt messages to a client that possesses the matching private key, and only the private key can decrypt the messages encrypted with the public key. Since only one person has the private key, only one person can decrypt your messages.

Conversely, a sender can encrypt data using the private key. The recipients of this message use the public key to decrypt the message. If the decryption is successful, the recipient can be sure that the message was encrypted with the corresponding private key. In this case many people can decrypt the message, but only the holder of the private key could have generated the message.

Authentication Requirements

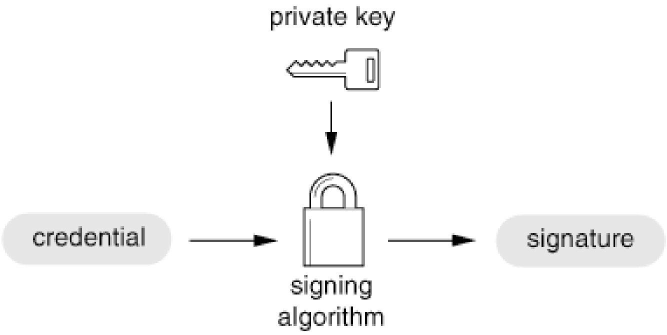
Before you can authenticate to eDirectory, you need a credential and a signature. You can obtain these by logging in to the network.

The *credential* is a data structure made up of a validation period and other user identification information. The *signature* is the result of a private-key encryption of the credential data.

In response to a login request, authentication services sends the client agent a private key that has been encrypted and stored in eDirectory. The client agent receives the password from the user. Nothing more is requested of the user from this point on. The client agent decrypts the private key.

The client agent then creates the credential with data unique to this session and encrypts it with the private key, thus creating the signature. Once the signature is created, the private key is erased from memory, but the signature and credential are retained. The following figure shows the principal items used to form the signature.

Figure 3-1 *Items Used to Form a Signature*



In many systems, a signature is itself the authentication mechanism and accompanies a message to its destination. This usage of a signature typically enables the recipient to verify that the message originated from its purported sender, by virtue of the fact that the receiver can decrypt it.

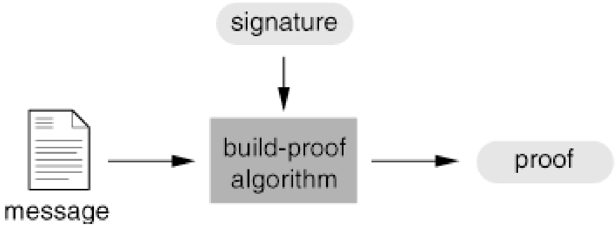
However, Authentication Services extends the concept of a signature further by using proofs. A proof is data derived from both the signature and the message. It is transmitted on the network with the request or message. Its use adds an extra measure of security by keeping the signature itself off the network.

Authentication Process

After constructing the credential and the signature, the client agent can authenticate itself to an eDirectory server on the network.

The client agent's request for client-server authentication is accompanied by a proof, which is constructed from values derived from both the signature and the request data (message) itself. The following figure shows the principal items used to construct the proof.

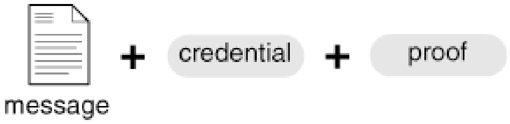
Figure 3-2 *Items Used to Form a Proof*



A proof is a part of all authentication dialogues. A proof ties the clients' signature to both the current request (message) and the current session, thus making each proof unique to the request it accompanies. The proof also makes it unnecessary to transmit the signature.

The request for authentication is transmitted to Authentication Services (a part of eDirectory) along with the proof and the unencrypted credential (see the following figure.) Then Authentication Services mathematically verifies that the proof is valid. The proof always assures the recipient that the message has not been modified since it was sent.

Figure 3-3 *Items Used to Confirm Authentication*

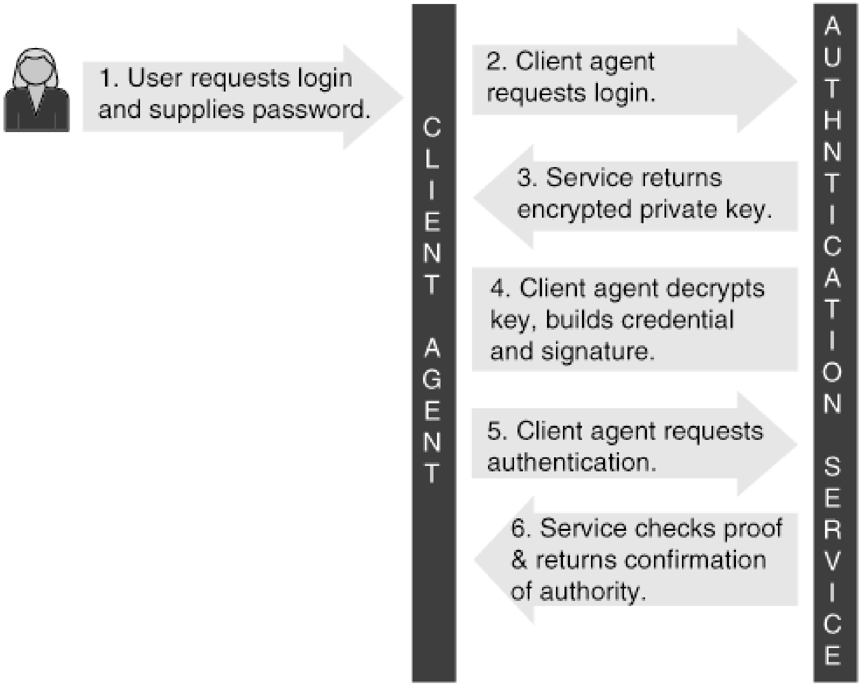


To summarize, a valid authentication provides the following guarantees:

- Only the purported sender could have constructed the message.
- The message came from the workstation where the authentication data was created.
- The message pertains to the current session between client and server.
- The message contains no information counterfeited from another session.
- The message has not been tampered with or corrupted.

The following figure summarizes the steps involved in initial authentication.

Figure 3-4 *Logging in and Authentication Summary*


1. User requests login and supplies password.
CLIENT AGENT
2. Client agent requests login.
3. Service returns encrypted private key.
4. Client agent decrypts key, builds credential and signature.
5. Client agent requests authentication.
6. Service checks proof & returns confirmation of authority.
AUTHENTICATION SERVICE

Background Authentication

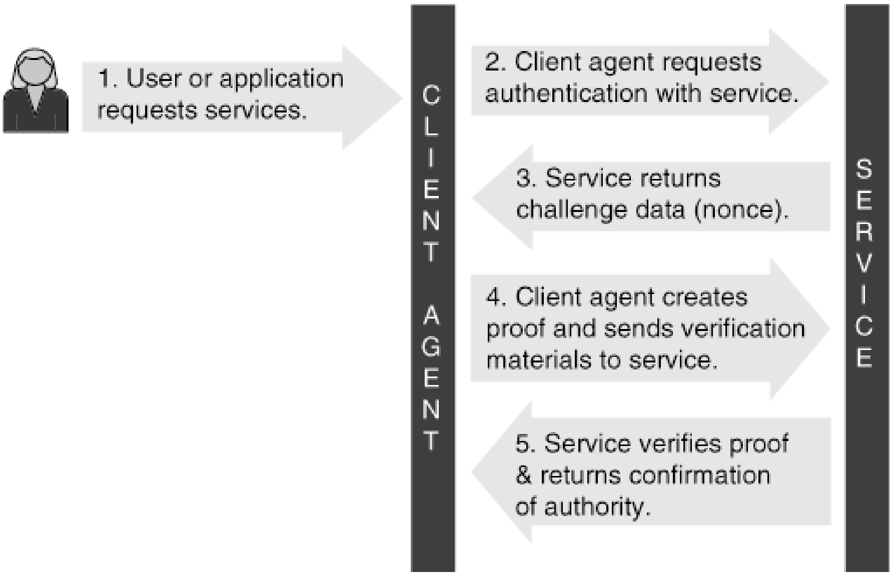
Background authentication with other participating services on the network can be accomplished in a manner similar to the initialization process. In background authentication, the client agent already has the authentication materials (credential and signature) on hand, and wants to authenticate itself to a new service. For example, the client agent may be seeking an attachment to a server. Since the client agent has already acquired the authentication materials, it can perform the authentication to the server without disturbing the user.

The client agent begins by sending a request to the desired service. In response, the service sends the client agent a challenge, or nonce. The nonce is a random number generated for the current transaction only. The same nonce is not used again.

The client agent computes a proof of the credential and nonce using the signature, and then sends the nonce, the credential, and the proof to the service.

The service then verifies that the proof was legitimately generated from the nonce and credential. This establishes the client's authenticity. The nonce ensures that the message was created for the current request and is not data from another session. The service returns a confirmation. The following figure shows the exchange that occurs during background authentication.

Figure 3-5 *Background Authentication Summary*


1. User or application requests services.
CLIENT AGENT
2. Client agent requests authentication with service.
3. Service returns challenge data (nonce).
4. Client agent creates proof and sends verification materials to service.
5. Service verifies proof & returns confirmation of authority.
SERVICE

Password Changing in eDirectory

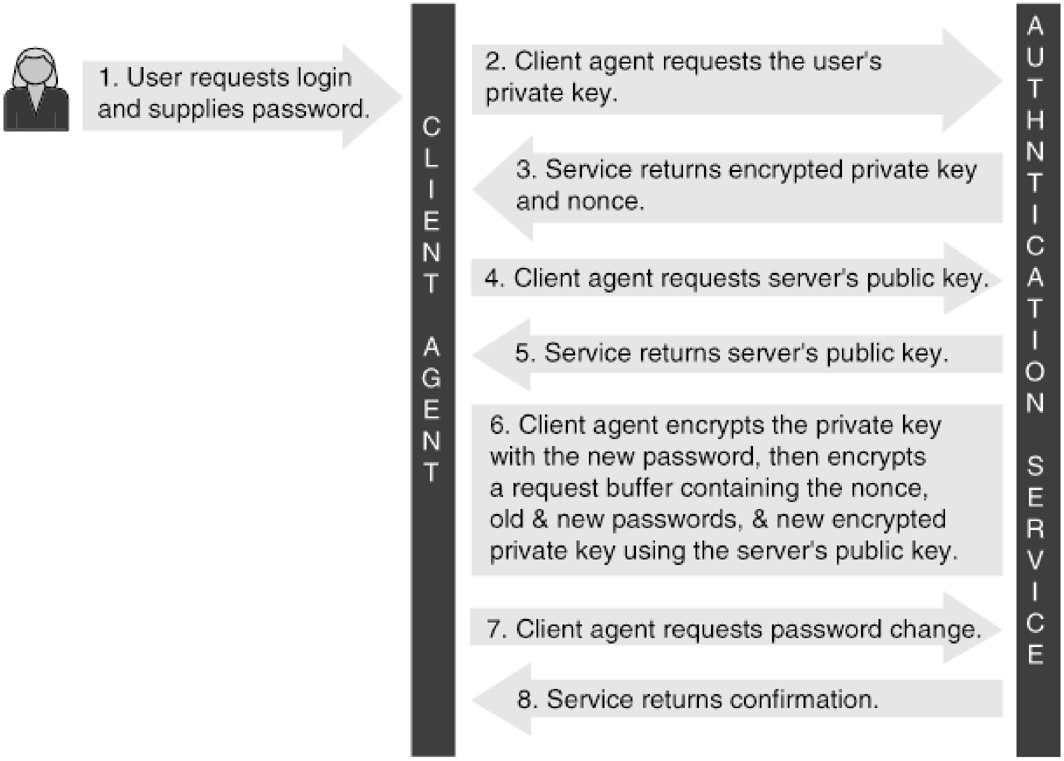
A special exchange is required in order for a client agent to change an object's password. Changing a password involves the client agent replacing the current encryption of the private key. The new private key encryption is generated with the new password. The generation and encryption of the private key occurs at the workstation, but the encrypted key is stored with eDirectory. It is not stored at the user's workstation.

The exchange begins with the user supplying the old and new passwords to the client agent. The client agent then asks eDirectory for the server's public key, and it is returned to the client agent. The agent then asks for the user's private key. The private key is returned from eDirectory with an accompanying nonce.

The client agent then encrypts the private key with the new password. The nonce, the old and new passwords, and the new private key encryption are placed into a message buffer that is encrypted using the server's public key, so that the combined data can be safely transmitted to the eDirectory Server and decrypted by the server using the server's matching private key.

When the client agent sends the request to change the password, the server is able to decrypt the message containing the nonce, the old and new passwords, and the user's newly encrypted private key. This is sufficient verification that the transaction is valid, so the server completes the password change and sends a confirmation of the password change back to the client agent. The following figure shows the steps required to change the password.

Figure 3-6 *Change Password Summary*



1. User requests login and supplies password.
CLIENT AGENT
2. Client agent requests the user's private key.
3. Service returns encrypted private key and nonce.
4. Client agent requests server's public key.
5. Service returns server's public key.
6. Client agent encrypts the private key with the new password, then encrypts a request buffer containing the nonce, old & new passwords, & new encrypted private key using the server's public key.
7. Client agent requests password change.
8. Service returns confirmation.
AUTHNTICATION SERVICE

Changing Public and Private Key Pairs

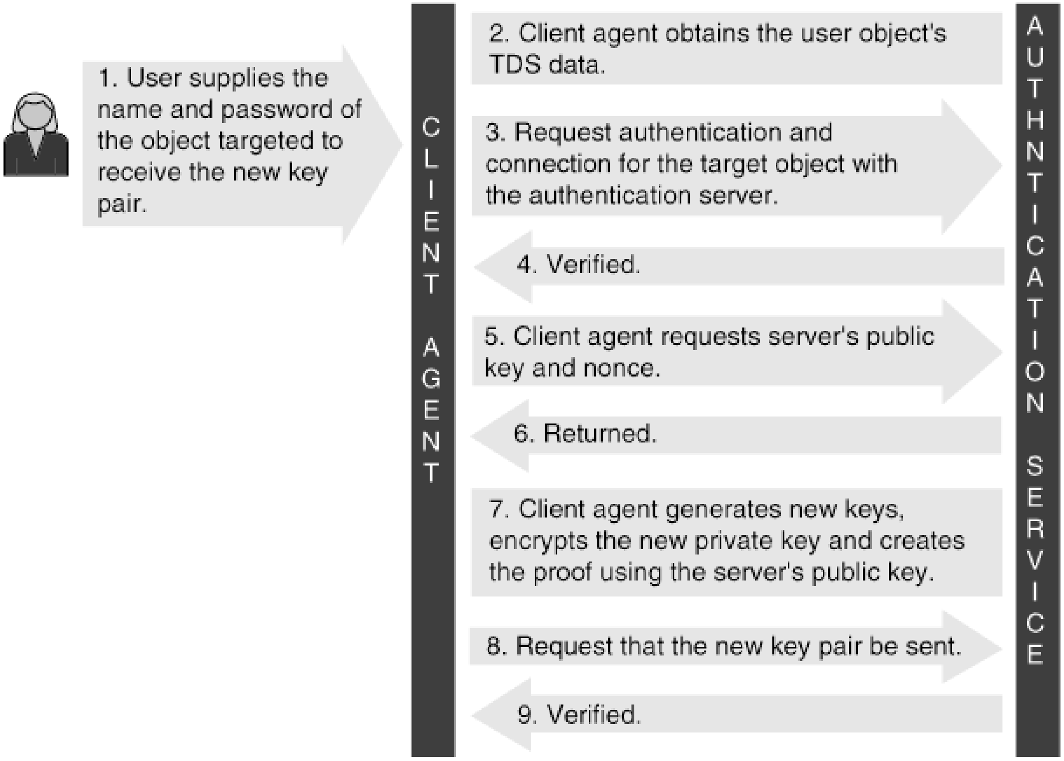
Each object in eDirectory has its own private/public key pair that is generated at the time the object is created. If conditions warrant it, a key pair can be changed. Setting or changing the public/private key pair for an object requires a special exchange between the client agent and Authentication. The exchange assumes that the client desiring to make the change has already authenticated itself to eDirectory. The client desiring to make the key pair change must have sufficient access privileges sufficient to change the values of the target object’s Private Key and Public Key attributes.

The exchange begins once the user gives the name and password of the object that will receive the new key pair. The object’s Tag Data Store (TDS) information is obtained. The TDS consists of a certificate, credential, and signature. The certificate consists of the name of the user object, the user’s public key and other information. The credential and signature have been discussed previously.

The client agent checks to see whether the object is currently connected and authenticated with the server performing the key pair update. If it is not, then an authentication is established. Next, the server’s public key and a nonce are obtained. The servers public key is used later to encrypt the key pair update request buffer prior to requesting that the pair be set.

The client agent then generates a new key pair with the intent of submitting it as the key pair for the target object. The new private key (of the new key pair) is encrypted with the target object’s user password, and is included in the request data used to create a proof. The data for the proof includes the nonce, the object’s ID, the password, the new encrypted private key, and the new public key. The server public key obtained earlier is used to encrypt this proof. The final step is to submit the request to set the new key pair for the target object. The following figure shows the steps required to set the object's key pair.

Figure 3-7 *Set Key Pair Summary*


1. User supplies the name and password of the object targeted to receive the new key pair.
CLIENT AGENT
2. Client agent obtains the user object's TDS data.
3. Request authentication and connection for the target object with the authentication server.
4. Verified.
5. Client agent requests server's public key and nonce.
6. Returned.
7. Client agent generates new keys, encrypts the new private key and creates the proof using the server's public key.
8. Request that the new key pair be sent.
9. Verified.
AUTHENTICATION SERVICE

3.1.2 LDAP Authentication

The LDAP v3 specification provides for three types of authenticated connections, or bindings:

- Anonymous bind—a no-name, no-password-required binding. For eDirectory, this is a connection without authentication, or simply an attachment to a server in the eDirectory tree. The LDAP user has access to the information granted to [Public].
- Clear-text password bind—a name and password binding, but the password must be clear text. The eDirectory server must be configured to accept unencrypted passwords for this type of LDAP binding to succeed.
- Secure bind—a name and encrypted password. LDAP supports a number of secure mechanisms (SASL and SSL). eDirectory versions previous to 8.7 support only SSL.

An LDAP connection grants rights only to the eDirectory tree. LDAP currently does not define a mechanism to support access to other network resources, such as a server's file system.

3.2 Access Control Lists

Authentication services are able to confirm a user's identity and validate that identity for other services. eDirectory uses that identity to control access to eDirectory information and uses access control lists to limit a user's rights to create, read, or modify a piece of eDirectory information.

A user that has not been authenticated has some access to the network because eDirectory confers public status on unauthenticated users. By default, these users have access to the information required to obtain a licensed connection to the server. However, the network administrator can grant additional rights to the user [Public], whose rights are given to all unauthenticated users.

The access control list (ACL) is the key component in controlling access to eDirectory information. The ACL determines which operations an entry can perform on another object and its attributes. In the eDirectory schema, the ACL attribute is a multivalued attribute type assigned as an optional attribute to the object class Top. Since all object classes inherit the characteristics of Top, all objects may have an ACL attribute.

The ACL attribute is an attribute on the object that is being accessed. Each assignment in the ACL attribute lists which entry has been granted rights to the object, what type of rights the entry has been granted, and what rights are being granted. When an entry has been granted rights to another object, the entry is called a trustee of that object.

This section describes the following characteristics of ACLs:

- "ACL Components" on page 43
- "ACL Operations" on page 44
- "Object Rights" on page 45
- "Attribute Rights" on page 45
- "Trustees" on page 46
- "Access Privileges Required for an Operation" on page 47
- "Managing ACLs" on page 48

3.2.1 ACL Components

Each ACL contains a list of trustees for which access has been defined. Each entry in an ACL has the following fields:

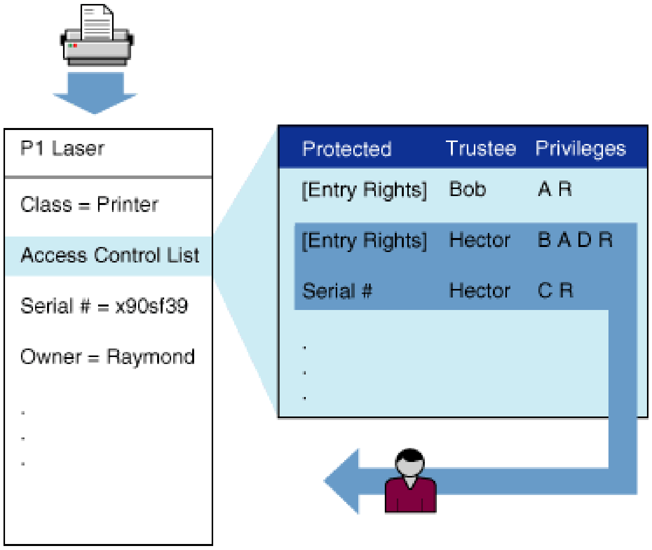
Field	Description
Trustee Name	This field is the complete name of the specific object in the eDirectory tree that is being granted rights. It can also be one of the following special entry names: [Inheritance Mask]—used to mask or filter privileges granted to an object. [Public]—used to grant rights to all entries in the eDirectory tree, even if the entry has not authenticated to the eDirectory tree. [Root]—used to grant rights to all authenticated entries. [Creator]—used to grant rights to the client that created the object. [Self]—used to allow objects to add or delete themselves as values of attributes.

Field	Description
Protected Attribute	This field specifies the type of right that is being granted. It can contain one of the following: The name of the attribute that the privilege set applies to—indicates that the rights apply just to this attribute [All Attributes Rights]—indicates that the rights apply to all the object's attributes. [Entry Rights]—indicates that the rights apply to the object that owns this ACL attribute.
Privilege Set	This field enumerates the set of privileges that have been granted to the subject. If [Inheritance Mask] is being specified, it enumerates the allowable privileges.

3.2.2 ACL Operations

The following figure shows how an access control list operates. In the figure, an ACL entry has been defined for a printer object. Attributes defined for the printer include the class, the ACL, the serial number, and the owner.

Figure 3-8 *An Access Control List Operation*



The ACL contains the names of the protected attribute, the trustee, and the privileges. Hector appears as the trustee two times in the list: once each for [Entry Rights] and Serial Number. At the object level ([Entry Rights]), Hector is assigned the Rename, Add, Browse, and Delete privileges. He is also assigned Compare and Read on the serial number attribute. These are his privileges regarding this particular printer as an eDirectory object. This access is to the eDirectory Information Base, not the printer or its queues. The printer and queue access are controlled by the printer’s access control mechanisms, not the ACLs of eDirectory.

3.2.3 Object Rights

Object rights give a trustee rights to the eDirectory object as a whole, not just to its attributes. When dealing with security, it is important to remember that there are two distinct parts of every object: the object itself, and its properties. The syntax used to denote rights to an object is [Entry Rights]. The rights a user (or any other object) might have to another object include Browse, Create, Delete, Inheritance Control, Rename, and Supervisor. These rights are described below.

Browse

Browse rights allow the user to “see” an object. If you were getting a list of all available objects, and you didn’t have Browse rights to a particular object, such as a Container, that container wouldn’t show up in your list. Your user object wouldn’t even know the object exists.

Create

Create rights allow the user to create subordinate objects to this object, when possible. For example, if the user has Create rights to a container, the user can add other objects inside of the container.

Delete

Delete rights allow the user to delete the object. A user cannot delete an object if that user doesn’t have right to all of the properties of the object. In other words, you must have Write rights to all of the properties of an object and Delete object rights in order to actually delete an object.

Inheritance Control

NetWare® 4.x does not support this right because in NetWare 4.x, rights granted to [Entry Rights] are always inherited.

In NetWare 5.x, Inheritance Control rights allow the user to control whether [Entry Rights] granted in an ACL are inherited. If inherited, the user can exercise the rights granted in the ACL on subordinate objects. NetWare 5.x allows you to either allow inheritance or block inheritance. (NetWare 5.x utilities and their documentation call this right Inheritable.)

Rename

Rename rights allow the user to rename an object.

Supervisor

If a user has Supervisor rights to an object, it is the same as having all of the other rights (Browse, Create, Delete, and Rename) in the [Object Rights] for that object.

3.2.4 Attribute Rights

Attribute rights allow access to the data associated with an object. (NetWare utilities and their documentation call attributes, properties, and attribute rights, property rights.)

A user (or other object) can be given rights to access specific pieces of data about an object, or rights to access all information of that object. For example, if User A needs to be able to see the name, addresses, and phone numbers of User B, you can grant User A rights to each of those attributes of User B. You can also grant User A rights to all of the attributes of an object. The syntax used to denote rights to all attributes of an object is [All Attributes Rights]. If rights are given to a specific attribute, the syntax used is merely the name of that attribute. Both [All Attributes Rights] and rights to specific attributes use the same rights in a bit mask.

These attribute rights are described below:

Compare

This right gives a user the ability to test the value of an object, but not read the value. In other words, the user cannot say, “What is your password?” but can ask, “Is this password correct?”

Read

This right gives a user the ability to see the values of the attributes that are not hidden attributes. By having Read rights, the user automatically has Compare rights.

Add or Delete Self

With this right, a user can add its own object as the value of the attribute. For example, if a user has Add or Delete Self rights to the Member property of an object, such as a mailing list, the user can add itself as a member of that group.

Write

Write rights give the user the ability to add, delete, and modify the value of an attribute, as long as that attribute is not a Read Only attribute. Having Write rights implies Add or Delete Self rights.

Supervisor

If the user has Supervisor rights to an attribute, it is the same as having all of the above attribute rights to that object.

Inheritance Control

NetWare 4.x does not support this right. In NetWare 4.x, inheritance of rights granted to [All Attribute Rights] is always allowed, and inheritance of rights granted to specific attributes is always blocked.

In NetWare 5.x, Inheritance Control rights control whether the user inherits the other rights granted to a specific attribute or to [All Attributes Rights]. The bit can be set to allow or to block inheritance on both [All Attributes Rights] and specific attributes.

In NetWare 5.x, this right enables the creation of managers who have rights to manage specific attributes, such as phone numbers, addresses, and passwords, without granting Supervisor rights to the objects. If the right is granted at the container level, the right can be inheritable to an entire branch of the eDirectory tree.

An important aspect of the privilege set is the relationship between the Read and Compare privileges. Compare is considered a subset of the Read privilege. If a subject has the Read privilege, it also has the Compare privilege, whether or not Compare has been explicitly assigned to the subject. However, having the Compare privilege alone does not grant the subject the Read privilege.

3.2.5 Trustees

The administrator can create access control lists (ACL) values to assign rights or privileges to an object or attributes. The object receiving the rights assignment is called a trustee. The assignments are known as trustee assignments, or more informally as ACLs. eDirectory uses these trustee assignments in conjunction with Section 3.3, “Inheritance,” on page 48 and Section 3.3.2, “Inheritance Masks,” on page 50 to compute the effective rights a given trustee has on a given object. The given object is the object whose ACL attribute contains the trustee assignment.

The trustee assignments are created as values of the multivalued ACL attribute attached to an eDirectory object. Trustee assignments are optional.

In NetWare® 4.x, trustee assignments made to objects flow down the eDirectory tree. For example if object X is given rights to a container object A that contains object B then X gets the same rights on object B that it has on object A. However, Object B could have an inheritance mask to filter some privileges. If so X will have only the rights to access B that pass through the filter.

NetWare 5.x adds new functionality. Trustee assignments to objects can flow down the eDirectory tree just as they did in NetWare 4.x and they can be blocked with inheritance masks. However, inheritance can also be disabled with the Inheritance Control right.

In NetWare 4.x, trustee assignments made for attributes do not flow down the tree unless the assignment is for all attributes. Assignments made to specific attributes do not apply to subordinate objects or attributes of those subordinate objects.

NetWare 5.x adds new functionality. The creator of the ACL determines whether the inheritance of rights to all attributes or to specific attributes is enabled or blocked. Because NetWare 5.x allows the inheritance of rights to specific attributes, network administrators can set up managers of specific eDirectory attributes, such as phone numbers, passwords, addresses, without granting Supervisor rights to the objects.

3.2.6 Access Privileges Required for an Operation

Since a trustee can have privileges to both an object and its attributes, the combination of these privileges may determine the operations available to the subject. Some operations on attributes require that privileges be assigned at both the object and the attribute levels.

Also, some operations (such as search, list, or move) can involve more than one object. It may be useful to take a closer look at the privileges involved in specific operations. The following table summarizes the privileges required at both the object and attribute level to perform a particular operation.

Table 3-1 *Required Access Privileges*

Operation	Object Privileges		Attribute Privileges
Compare attribute value	NONE	AND	Read or Compare
Read attribute value	NONE	AND	Read
List subordinates	Browse	AND	NONE
Add object	Add (on the parent object)	AND	NONE
Search	Browse on each object	AND	Compare on each attribute in filter; Read on each attribute returned.
Add attribute to object	NONE	AND	Write
Add value to attribute	NONE	AND	Write
Delete attribute	NONE	AND	Write
Delete value of attribute	NONE	AND	Write

Operation	Object Privileges		Attribute Privileges
Delete object	Delete	AND	Write on each present attribute
Move object	Delete (at the source location); Add (at the destination)	AND	Write on each present attribute
Write self	NONE	AND	Self
Modify Name (RDN)	Rename	AND	NONE

It should be noted that the Supervisor privilege on an object or attribute gives the subject all privileges allowing any of the functions to be performed. However, if the Supervisor privilege is inherited, it can be restricted by an inheritance mask.

3.2.7 Managing ACLs

ACL values can be used to allow objects to change the ACL itself. Objects given certain privileges to the ACL can access it and change the rights within it. The privileges assigned in the ACL can be not only for the ACL but for the object of the ACL and all other attributes of that object.

Be careful in granting the Write, Self, and Supervisor privileges. Supervisor gives all privileges, Write allows an object to change the ACL values, and Self allows an object to add and delete itself to the value.

The Self privilege can be granted to ease the burden of managing certain attributes of objects. Giving objects the ability to add themselves to certain lists (attributes) of objects eliminates the need for a manager to add and delete those objects. An example would be a distribution list. If you made a multivalued attribute called Party Group and gave [Root] the Self privilege to that attribute, objects wishing to join your Party Group could add themselves to the group. No one could delete others from the group, but they could add or delete their own object as desired.You would want to maintain all privileges to the Party Group attribute yourself so you could read, add and delete any object at any time. This would allow you to choose to let the individuals manage their membership in the group, or manage it yourself.

3.3 Inheritance

ACL inheritance determines whether the rights granted to a parent object can flow down the eDirectory tree to subordinate objects. Access control privileges are applied according to the hierarchical structure of eDirectory. For example, if a subject has been granted the Browse privilege for an object, the subject will also have the privilege to browse that object’s subordinates.

In NetWare® 4.x, when a user is given [Entry Rights] or [All Attributes Rights] to a container object, the user is given the same rights to all of the subordinated objects of that container. These rights are called inherited rights because they are not explicitly given to the subordinate objects, but are received (or inherited) from the container object.

If the user is given specific property rights, and not [All Attributes Rights], those rights do not flow down to the subordinate objects of the container. In other words, if the user has Read rights to the Name attribute of a container, the user does not have Read rights to the Name attribute of the objects subordinate to the container. However, it the user has [All Attributes Rights] to the container, the user has [All Attributes Rights] to all objects subordinate to that container as well.

In NetWare 5.x, inherited rights can behave as they did in NetWare 4.x, but NetWare 5.x allows them to be configurable. When a user is given [Entry Rights] or [All Attributes Rights] to a container object, the Inheritance Control right, a new right in NetWare 5.x, determines whether the user is given the same rights to the subordinate objects of that container. Also this right determines whether the rights to a specific attribute can be inherited. Both the blocking of inheritance of [Entry Rights] or [All Attributes Rights] inheritance and the enabling of inheritance for specific attributes is new functionality in NetWare 5.x.

The inheritance of ACLs to specific attributes enables the creation of managers who can administer certain attributes in a branch of the eDirectory tree without having to grant the managers Supervisor rights to the objects. For example, you can create a telephone number manager by granting the manager an ACL to the Telephone Number attribute of an eDirectory container object. The privilege set should include the Read, Write, and Inheritance Control rights.

Inheritance is influenced by

- "Equivalence" on page 49
- "Inheritance Masks" on page 50
- "Inherited ACLs" on page 51
- "Effective Rights Calculations" on page 52

3.3.1 Equivalence

Security equivalence simply means that a user has the same rights as another object in the eDirectory tree. Logged in users are always security equivalent to the root-most object in the tree, [Public], and all the parent objects in their distinguished name.

One method for creating a security equivalence is to make one (principal) object a member of another (secondary) object's Security Equals attribute list. Access to a protected object can then be granted to the principal object by granting privileges to the secondary object. If the principal is also granted explicit access to the object, then the access privileges are the sum of the two privilege sets. The resulting privileges after all inheritance filters are applied are called the principal's effective privileges (or effective rights). A principal object will receive equivalences from as many objects as it is security equivalent to.

An object can be made security equivalent to a group object. The NetWare® utilities make an object security equivalent to any group the object is made a member of. Simply being a member of a group, however, does not give an object privileges of the group. It is having the group in the object's security equals list that makes the object security equivalent.

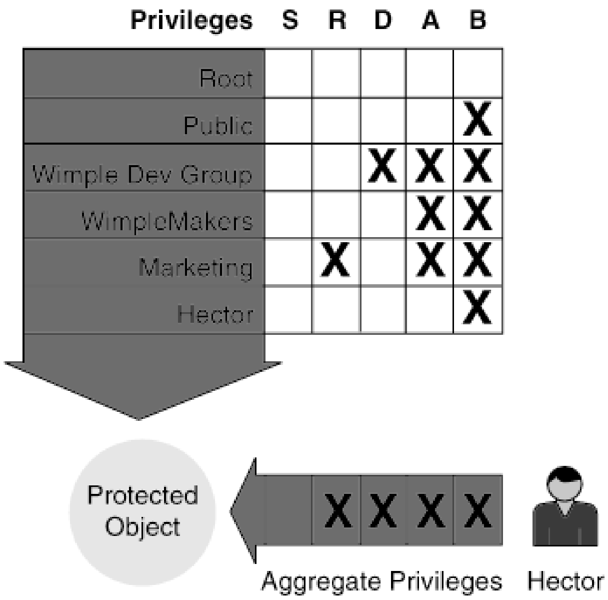
Security equivalences are also formed by virtue of a trustee being a member of a particular subtree. All objects in the tree are equivalent to their superiors. Any privileges assigned to a parent object will be inherited by the subordinate objects.

Security privileges can be assigned all objects in the tree by granting privileges to the root-most object in the tree. This method uses the name of the root-most object as the trustee of the privilege set. All objects are subordinate to the root-most object by default so any privileges assigned to that object are applied to all objects. Similarly, all NetWare 4.x and 5.x objects are implicitly equivalent to [Public]. Even users not logged in to the network (not authenticated) can be given privileges to objects in eDirectory by adding an ACL entry with [Public] as the trustee of a privilege set.

The following figure shows how the accumulated privileges for a user are determined. The user's distinguished name is Hector.WimpleMakers.Marketing, and he is the equivalent of 4 objects:

[Public], Wimple Dev Group, WimpleMakers, and Marketing. He is equivalent to WimpleMakers and Marketing because he is subordinate to these objects in the eDirectory tree. He is equivalent to [Public] because he is attached to the eDirectory tree, to the root-most object in the tree because he is authenticated to the eDirectory tree, and to Wimple Dev Group because his Security Equivalents attribute includes this group. Even though no privileges are assigned explicitly to Hector, his aggregate privileges are the sum of all of his equivalences. Hector's effective privileges will be his aggregate privileges as long as no inheritance masks restrict them.

Figure 3-9 *Accumulating Privileges*



See Also:

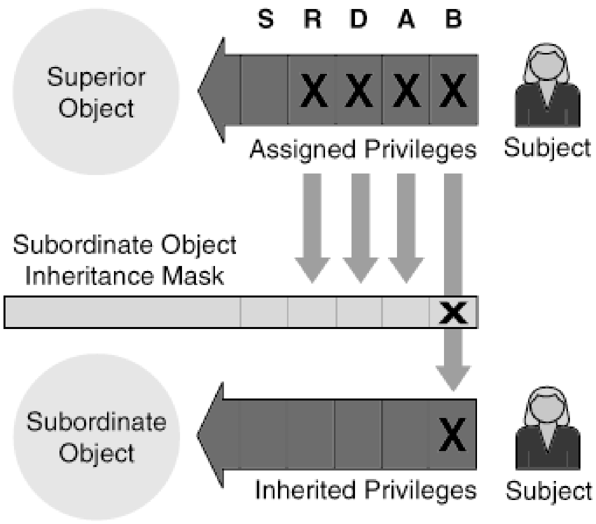
- Section 3.3, "Inheritance," on page 48
- Section 3.2, "Access Control Lists," on page 42

3.3.2 Inheritance Masks

Inheritance can also be restricted by an inheritance mask. An inheritance mask is an entry in the ACL whose subject field contains a special inheritance mask name. The privilege set in the inheritance mask lists those privileges that may be granted through inheritance. Inheritance masks can be defined at both the object level and the attribute level. Inheritance at the object level and the attribute level are kept separate.

The following figure shows how inheritance operates. In this figure, a subject is assigned R, D, A, and B privileges to an object that is the superior of another object. The subject has not been assigned privileges to the subordinate object. By inheritance, the subject would normally have the same rights on the subordinate object. The subordinate object, however, has an inheritance mask that allows only B to be inherited. As a result, the R, D, and A privileges are not allowed and only the masked privilege is inherited.

Figure 3-10 *Inheritance with a Inheritance Mask*



An inheritance mask applies only to inherited rights. In NetWare 4.x, inherited rights are assignments made of objects and all attributes. In NetWare 5.x, inherited rights are determined by the setting of the Inheritance Control right.

If a subject receives privileges to a protected object by explicit values in the ACL, inheritance does not apply and the inheritance mask is ignored. However, where no ACL value is defined for a subject, the inheritance mask indicates which privileges may be inherited.

Inheritance masks are used to selectively filter privileges down the eDirectory tree. For example, suppose user Joe is give [Entry Rights] of Browse, Create, and Rename to a container. This particular container has four subordinate container objects, each of which has other subordinate objects. You want Joe to inherit his rights from the first container down to all the subcontainers except one. In this one subcontainer, you don’t want Joe to have the right to create subordinate objects. An inheritance masks easily solves this problems. You can place an inheritance mask on the subcontainer that filters out the Create [Entry Right]. All objects that have the Create [Entry Rights] to the parent of this container do not inherit the Create right to this subcontainer.

Inheritance masks are stored as ACL values of eDirectory objects. eDirectory uses this mask to compute the effective rights one object has on another object or attribute. If no inheritance mask exists, all inherited privileges are granted.

Inheritance masks can be defined for objects, specific attributes, or all attributes of an object. In some Novell literature these masks are referred to as Inherited Rights Filters (IRFs). When programmers request creation of one of these masks they must specify [Inheritance Mask] as the trustee name of the ACL entry. The privilege set specifies the rights that can be inherited. The protected attribute specifies [Entry Rights], [All Attributes Rights], or a specific attribute.

3.3.3 Inherited ACLs

Because of eDirectory inheritance, access control information defined at one point in the eDirectory tree is applied to all subordinate regions of the tree. The effect of inheritance is cumulative. If a subject receives privileges as a user at one place in the tree and as a group member at another point in the tree, the sum of these privileges is available to the subject at lower points in the tree. (Inheritance may be limited by an inheritance mask.)

However, eDirectory partitioning creates gaps in the flow of access control information downward through the tree. It would be inconvenient to try to assess a subject's inherited privileges across several partitions for each request that the subject initiated on a particular partition. eDirectory alleviates this problem through Inherited ACLs.

The Inherited ACL is an attribute attached to each partition root object. Like a regular object ACL, each entry in the Inherited ACL has a subject field, a protected attribute field, and a privileges field. The Inherited ACL has an entry for every subject for which privileges have been defined in the superior partitions.

When a name server is calculating the effective rights of a subject in relation to a protected object, the name server begins with the Inherited ACL. To any privileges found in the Inherited ACL, the name server adds any additional privileges found in the given partition leading down to the protected object.

The Inherited ACLs are maintained by an eDirectory background process. When ACL modifications are made, this process initiates a summarization of the ACL for any replicas that are affected by the change and that are stored on the server. The result of this summarization is forwarded to any subordinate partition object. Within the partition, inherited ACL information is propagated among replicas by the partition's synchronization processes.

Inherited ACLs are automatically created by the eDirectory server at the time a new partition is created.

3.3.4 Effective Rights Calculations

eDirectory uses everything described in the sections titled "Object Rights" on page 45, "Attribute Rights" on page 45, Section 3.3, "Inheritance," on page 48, and Section 3.3.1, "Equivalence," on page 49 to calculate the Effective Rights of an object to another object. Inheritance masks can mask out rights that are inherited from higher in the tree, but may be overridden by explicitly assigned rights and security equivalences below the inheritance mask. You need to be familiar with how each of the rights-assignment methods work before you can design an eDirectory tree with the security implemented as you want it, and without confusion as to why a user does or does not have the rights you desired.

3.4 NetWare File System

The access controls and restrictions for eDirectory are similar in concept to those used for access to the NetWare® file system. eDirectory identifies the volumes the file system stores, but the file system enforces its own access controls.

In order to access a specific server's file system you must:

- Have eDirectory privileges to see and locate the server and volume
- Establish a licensed connection to the server.
- Have file system access privileges

To locate the part of the file system you want to access you must have the ability to find the Volume object for the volume (for example, SYS:) that you wish to access. Each volume on all servers in the network will be represented by an object in eDirectory. If you have eDirectory privileges to see and read a particular volume object then you will be able to locate the server on which that volume resides and attempt to access that volume.

Having located the server and volume you wish to access, you must then obtain a licensed connection to the server. Each server has a limited number of licensed connections it can handle. An object obtains one of those connections by first authenticating itself through eDirectory, then requesting a connection.

If you have Write or Supervisor privileges on a server object in eDirectory, you automatically have Supervisor privileges on that server's file system. Otherwise your privileges to a NetWare server are according to the trustee assignments found in the given server's file system security mechanism.

The file system trustee assignments for NetWare 4.x and 5.x are the same as for previous versions of the operating system. These trustee assignments are kept with the volume with which they are associated and are not part of the eDirectory database. A file system trustee assignment consists of the granted rights and an Entry ID for the trustee. eDirectory supplies the Entry ID for the trustees. This Entry ID is a unique ID for the trustee and is valid only on the local server. If the server does not contain the partition which contains the entry's object, eDirectory creates an external reference on the server and this external reference supplies an Entry ID for the file system.

If you have adequate trustee assignments, your request to allocate a file system directory handle will be granted. At that point you will be able to access everything in that area of the file system according to your effective (file system) rights.

Partitions and Replicas

4

Novell® eDirectory™ is a distributed database, a group of features which supports storing portions of the eDirectory database on multiple servers and provides fault tolerance by distributing multiple copies of the same information across the network. These features include

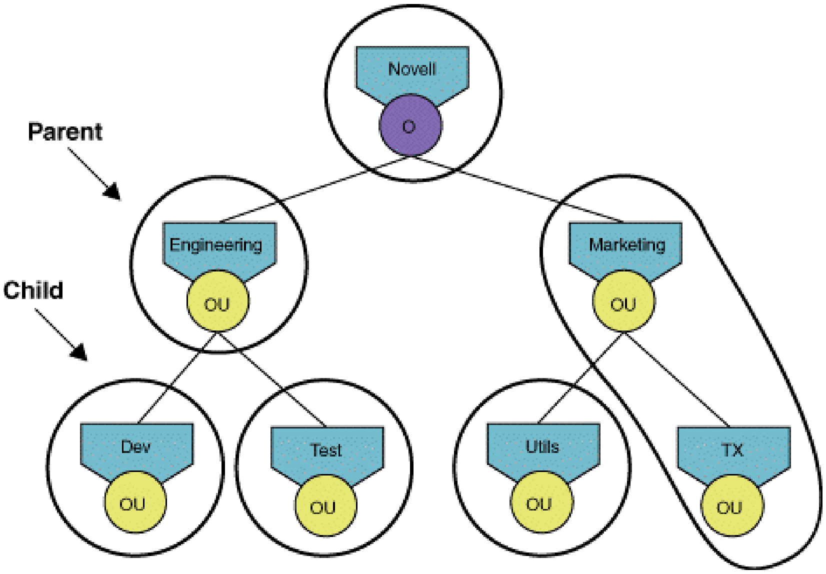
- Partitioning
- Replication
- Distributed reference management
- Partition and replica management
- Background synchronization processes

4.1 Partitioning

eDirectory divides the tree into logical subtrees called partitions. Although any part of the tree can be considered a subtree, a partition forms a distinct unit of data for storing and replicating eDirectory information. Partition boundaries cannot overlap, because each entry in the eDirectory tree must appear in only one partition.

The figure below illustrates a partitioned tree with Novell as the tree's root partition.

Figure 4-1 *Partitioned eDirectory Tree*



A partition subordinate to another partition is called a child partition, while its immediate superior is called a parent partition. Thus, engineering.novell is a parent partition and dev.engineering.novell and test.engineering.novell are child partitions.Even though Figure 4-1 on page 55 shows the eDirectory tree partitioned into six partitions, end users browsing the tree see the tree as one tree and are not aware of crossing any boundaries when they move from one partition to another.

The Tree Root object is not the same as a partition root. The Tree Root object is the topmost object of the entire eDirectory tree. A partition root is the root-most object in the partition and from which the partition takes its name. In Figure 4-1 on page 55, the six partitions have the following names: Engineering, Dev, Test, Marketing, and Utils. The top or root partition of the tree can be called either Novell or [Root].

An eDirectory server can store multiple partitions or none. eDirectory does not require that a server hold a partition. There are no restrictions placed on the partitions that can be stored together on the same eDirectory server. For example, none of the partitions need be contiguous to each other. For example using the eDirectory tree in the Figure above, a server could hold the Engineering and Utils partitions.

A partition's root object stores system information for maintaining the partition. A partition root object can be any container class such as Locality, Organization, and Organizational Unit. Partition attributes are assigned to the object along with the regular attributes defined by the object's base class.

4.1.1 Partition Class

The Partition object class has no containment classes defined, nor does it have any naming attributes. Top is its only super class. The Partition class does have mandatory and optional attributes. Some of this attribute information is specific to the local replica and some is data shared by all the partition's replicas.

Mandatory Partition Attributes. The following table lists attributes that are mandatory for partition objects.

Table 4-1 *Mandatory Partition Attributes*

Attribute	Comment
Convergence	This attribute determines how frequently the partition attempts to synchronize its replicas. A low value indicates that the partition should wait until the next scheduled synchronization interval to send updates. A high value indicates that the partition should send the information immediately (within 10 seconds).
Partition Creation Time	This attribute is an identifier for the partition's current set of replicas.
Replica	This attribute stores a list of servers that store replicas of the partition. Each entry includes the server name, the replica type, the replica number, and "best guess" network address for the server. This list is called the replica list, and the servers that are in the list make up the replica ring.

Optional Partition Attributes. The following table lists attributes that are optional for partition objects.

Table 4-2 *Optional Parition Attributes*

Attribute	**Comment**
High Convergence Sync Interval	This attribute specifies the interval (in seconds) at which a partition synchronization will occur if no intervening events have caused synchronization and the Convergence attribute is set to high.
Inherited ACL	This attribute stores a summary of access rights assigned within the partition's superior partition.
Low Convergence Sync Interval	This attribute specifies the amount of time (in seconds) that must pass from the start of one partition synchronization to the next if the Convergence attribute is set to Low.
Received Up To	This attribute stores a time stamp for the last time the replica received an update.
Synchronized Up To	This attribute is a list of time stamps indicating the last time all servers holding a replica of the specified partition were synchronized.

4.2 Replication

Replication adds fault tolerance to the database because replication allows the database to have more than one copy of its information.A single instance of a partition is called a replica. Partitions can have multiple replicas, but only one replica of a particular partition can exist on each server. Servers can hold more than one replica, as long as each replica is of a different partition.

Replica management is based on

- Replica Types
- Replica Lists or Rings

4.2.1 Replica Types

Replicas must be designates as one of six types:

- Master
- Read/write
- Read-only
- Subordinate reference
- Filtered read/write replica
- Filtered read/only replica

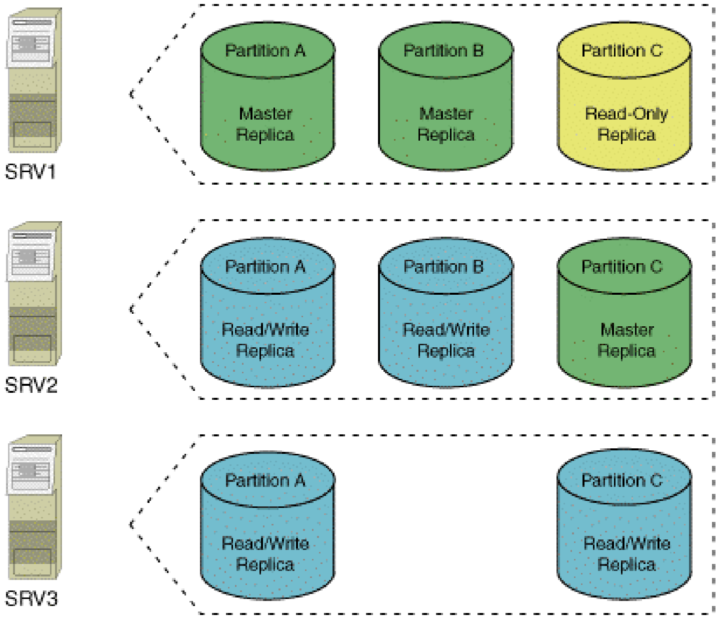
The initial partition replica is called the master replica. A partition can have only one master replica; additional replicas are either secondary or read-only replicas. There is no restriction on the number of these replicas.

Each replica stores the name of the replica's server, the replica type (master, secondary, or read-only), and the best known network address for the server. A replica also stores a list of replica pointers that identifies all the replicas of that partition. The replica relies on the list of replica pointers to spread updates throughout the partition.

Both master and secondary replicas can be used to create, modify, and delete objects; however, only the master replica can be used to create and delete subordinate partitions. (The master replica is also required for the move operation and for all partition operations.) A read-only replica can be used only to read the information in the partition; it cannot be used to write to the partition.

The following figure shows three partitions (A, B, and C) replicated across three name servers (SRV1, SRV2, and SRV3). NS1 stores the master replicas of partitions A and B and a read-only replica of partition C. SRV2 stores the master replica of partition C and secondary replicas of A and B. SRV3 stores secondary replicas of A and C. Given this arrangement, a request to add an object to partition A could be sent to any of the servers. A similar request for partition B could only be handled by SRV1 or SRV2, and SRV2 or SRV3 could handle such a request for partition C.

Figure 4-2 *Replication of Partitions*



To create a new partition subordinate to Partition A or B, the request must be directed to SRV1. To create a new partition subordinate to Partition C, the request must be directed to SRV2.

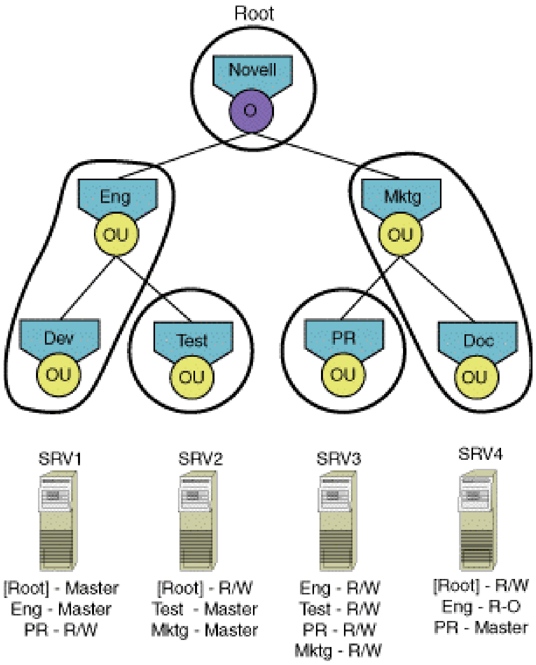
All three servers can handle requests to add or modify an object on Partition A. SRV1 and SRV2 can handle similar requests for objects on Partition B, and only SRV2 and SRV3 can handle similar requests for objects on Partition C.

4.2.2 Subordinate Reference Replicas

Subordinate Reference replicas are quite different from master, read/write and read-only replicas which can be seen and manipulated by system administrators. Subordinate Reference replicas, however, are invisible to system administrators. eDirectory creates, manages, and uses them to provide tree connectivity.

Each subordinate reference is a complete copy of a given partition's root object but is not a copy of the whole partition. As a general rule, subordinate references are placed on servers that contain a replica of a parent partition but not all the child partitions. In other words, a subordinate reference points to an absent subordinate partition. In this case, the server contains a subordinate reference for each child partition it does not store.Subordinate references provide tree connectivity by referring to replicas the server may need to find. Because the subordinate reference is a copy of a partition root object, it holds the Replica attribute, which lists all the servers on which replicas of the child partition can be found. eDirectory uses this list to locate replicas of the subordinate partition. Figure 4-3 on page 59 shows a partitioned tree and the servers with their replicas.

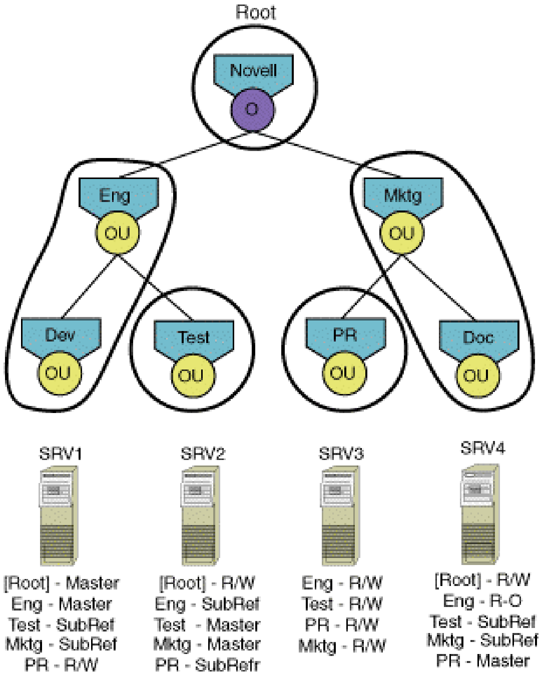
Figure 4-3 *Replica Placement in an eDirectory Tree*



Some of the servers in the figure above hold replicas of the parent but not replicas of all corresponding children. These servers must also hold subordinate references to the child partitions they do not hold. For example, because server SRV4 holds a replica of the Eng partition but not of Test, it must hold a subordinate reference to Test.

Figure 4-4 on page 60 displays the placement of the subordinate references. SRV1 requires subordinate references to mktg.novell and test.eng.novell because it holds replicas of the parent partitions but not of the child partitions.

Figure 4-4 *Replica Placement and Subordinate References*



On server SRV1, the subordinate reference of mktg.novell is a complete copy of the root object, mktg.novell, but not of its subordinate objects; the subordinate reference of test.eng.novell is a complete copy of the entire partition, since test.eng.novell is the only container object in the partition. Users cannot change a subordinate reference's replica type.Besides providing tree connectivity, subordinate references also help determine rights. Because a subordinate reference holds a copy of the partition root object, it holds that object's Inherited ACL attribute, which summarizes the Access Control Lists up to that point in the tree. See "Inherited ACLs" on page 51 for more information.

4.2.3 Filtered Replicas

Filtered replicas contain a filtered set of objects or object classes along with a filtered set of attributes and values for those objects. For example, you might want to create a set of filtered replicas on a single server that contain only User objects from various partitions in the eDirectory tree. In addition to this, you can choose to include only a subset of the User objects' data (for example,. Given Name, Surname, and Telephone Number).

A filtered replica can construct a view of eDirectory data onto a single server. To do this, filtered replicas let you create a scope and a filter. This results in an eDirectory server that can house a well-defined data set from many partitions in the tree.

The descriptions of the server's scope and data filters are stored in eDirectory and can be managed through the Server object in ConsoleOne.

A server hosting one of more filtered replicas has only a single replication filter. Therefore, all filtered replicas on the server contain the same subset of information from their respective partitions. The master partition replica of a filtered replica must be hosted on an eDirectory server running eDirectory eDirectory 8.5 or later.

Filtered replicas can:

- Reduce synchronization traffic to the server by reducing the amount of data that must be replicated from other servers.
- Reduce the number of events that must be filtered by DirXML.
- Reduce the size of the directory database.

  Each replica adds to the size of the database. By creating a filtered replica that contains only specific classes (instead of creating a full replica), you can reduce the size of your local database.

  For example, if your tree contains ten thousand objects but only a small percentage of those objects are Users, you could create a filtered replica containing only the User objects instead of a full replica containing all ten thousand objects.

Other than the ability to filter data stored in a local database, the filtered replica is like a normal eDirectory replica and it can be changed back to a full replica at any time.

4.2.4 Replica Lists or Rings

Each replica contains a list of servers that contain replicas of the partition. The replica list is stored in each replica as a Replica attribute of the partition's root-most container entry. This list provides information needed for navigating the eDirectory tree and synchronizing the replicas. The replica list contains the following elements for each replica:

- **Server Name.** The name of the server where the replica is located.
- **Replica Type.** The type of the replica stored on the server designated in the Server Name field. (The type is either Master, R/W, RO, or SR.)
- **Replica State.** The status of the replica. (The status modes include On, New, Replica dying, among others.)
- **Replica Number.** The number that the master assigned to this replica at the time the replica was created.
- **Network Address.** The server's address.
- **Remote ID.** The Entry ID of the replica's partition root entry.

The group of servers storing this information form the partition's replica ring.

In an eDirectory tree with NetWare 4.x, NetWare 5.x, and NDS 8 servers, the servers in the replica ring determine which eDirectory features are supported. For example, to use the new Inheritance Control right, all the servers in the replica ring must be NetWare 5.x servers. To use auxiliary classes, all servers in the replica ring must be NDS 8 servers.

Servers in a replica ring do not all have to use the same transport protocol. Some servers can be configured to use only IPX while others use only IP. The only requirement for multiple protocols in a replica ring is that at least one server in the replica ring must support multiple protocols. To prevent a single point of failure in a mixed protocol replica ring, more than one server in the ring should support multiple protocols.

4.3 Distributed Reference Management

Just as subordinate references help maintain connectivity between parent and child partitions stored on different servers, the following other kinds of references also help keep eDirectory trees connected and synchronized:

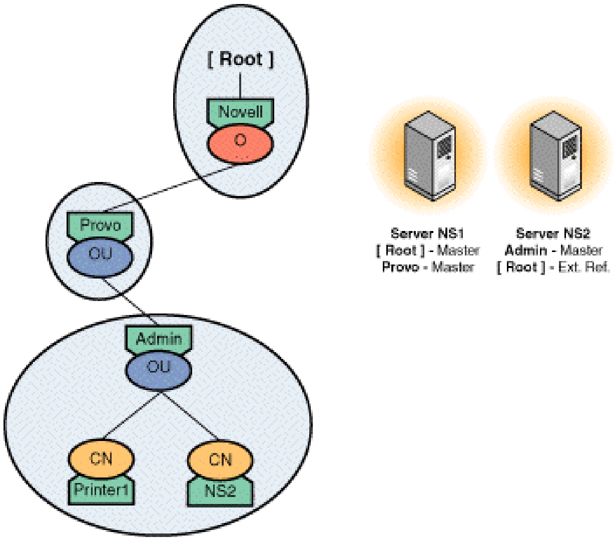
- External references
- Back links
- Distributed reference links (DRLs)
- Obituaries

4.3.1 External References

A server usually stores replicas of only some of a directory's partitions. Sometimes a server must hold information about entries in partitions that the server does not store. Often, the server requires information about an entry in a parent partition. At other times, the server requires information about entries in partitions that are not parents or children of partitions it stores. For example, the file system may need to refer to these entries because entries from diverse partitions have been granted rights to directories on this server.

In the following figure, server NS2 does not store a replica of the [Root] partition. To maintain connectivity with the tree root, server NS2 stores an external reference of [Root].

Figure 4-5 *Server Replicas with External References*



eDirectory stores these types of information in external references, which are place holders containing information about entries that the server does not hold. External references are not “real” entries because they do not contain complete entry information.

Besides providing connectivity, external references improve system performance by caching frequently accessed information. Currently, eDirectory caches only an entry's public key, which is stored as an attribute on the external reference.

Creating External References. eDirectory creates external references for the following operations:

- **Authentication.** A user authenticates to a server, and this user does not have an entry stored in a partition on the server. To enable authentication, the server must create an external reference so that an entry ID can be given to the authentication process.
- **Browsing.** When a user, browsing the eDirectory tree, requests information about an entry that is not stored locally, eDirectory creates an external reference to the entry.
- **Security Equivalence.** Users who authenticate to the server can have security equivalence to entries not stored locally. Such entries require external references.
- **Attributes of Local Entries.** Some attributes, such as Member, take a list of entries and can have entries of objects that are not stored locally. Each such entry requires an external reference.
- **File System.** The file system uses entry IDs to maintain a list of owners and trustees of files and directories. Trustees or owners that are not local entries require external references.

In addition, eDirectory creates external references when a replica is removed from the server. eDirectory changes all of the entries in the removed replica into external references and marks them as expired.

eDirectory uses the following rules when creating external references:

- eDirectory never creates an external reference below a real entry in the tree.
- eDirectory never creates a subordinate reference below an external reference in the tree. Any subordinate references below an external reference will be removed during synchronization.

Deleting External References. On each server, eDirectory deletes expired external references if they have not been used within a specified time period. The system administrator can use a SET parameter to set a number of days after which eDirectory deletes external references that have not used, are not needed for another entry's context, or do not contain information that the operating system needs.

To remove expired external references, eDirectory builds a list of unused external references by checking the life span interval of each external reference. This interval defaults to eight days and thirty minutes.

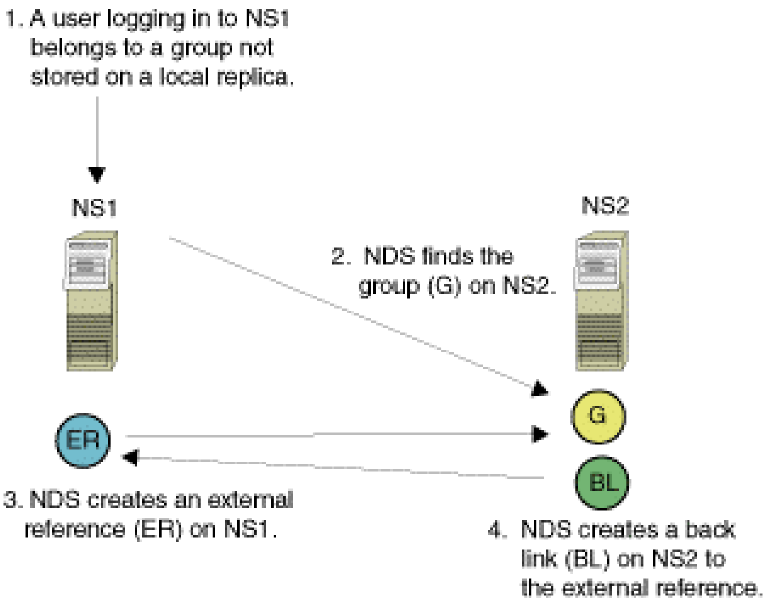
The back link process checks to see if the file system must access any of the external references. If the file system uses the expired external references, they are not taken off the delete list. The back link process deletes the remaining entries on the list. The janitor process is responsible for purging the deleted external references.

When eDirectory updates entries and partitions, it also must update external references created for those entries. Synchronizing external references is usually done by the server receiving the original synchronization request; however, any read/write replica can initiate synchronization if the external reference is being deleted or renamed.

4.3.2 Back Links

When eDirectory creates a new external reference for an entry not stored on the local server, eDirectory locates the non-local entry to which the external reference points. On that entry, it stores a Back Link attribute, which points back to the external reference. The back link maintains connectivity between the server holding the external reference and the server hold the object to which the external reference points. Figure 4-6 on page 64 illustrates this process.

Figure 4-6 *Back Links and External References*


1. A user logging in to NS1 belongs to a group not stored on a local replica.
NS1
NS2
2. NDS finds the group (G) on NS2.
G
ER
BL
3. NDS creates an external reference (ER) on NS1.
4. NDS creates a back link (BL) on NS2 to the external reference.

If eDirectory can't create the back link, it retries 9 times. The default retry interval is currently 3 minutes. If eDirectory cannot create the back link within 3 minutes, the task is assigned to the back link process which executes on a time interval set by the system administrator. The current default interval is 25 hours.

eDirectory uses back links to update external references in cases where the real object has been renamed or deleted. The back link process has two basic functions:

- Removes any expired or unnecessary external references from the system.
- Creates and maintains any back links that eDirectory could not create when it created the external reference.

When eDirectory removes an external reference, the back link to that external reference must be deleted. The server holding the external reference requests that the server holding the real entry delete the back link, and the server holding the back link then deletes the reference.

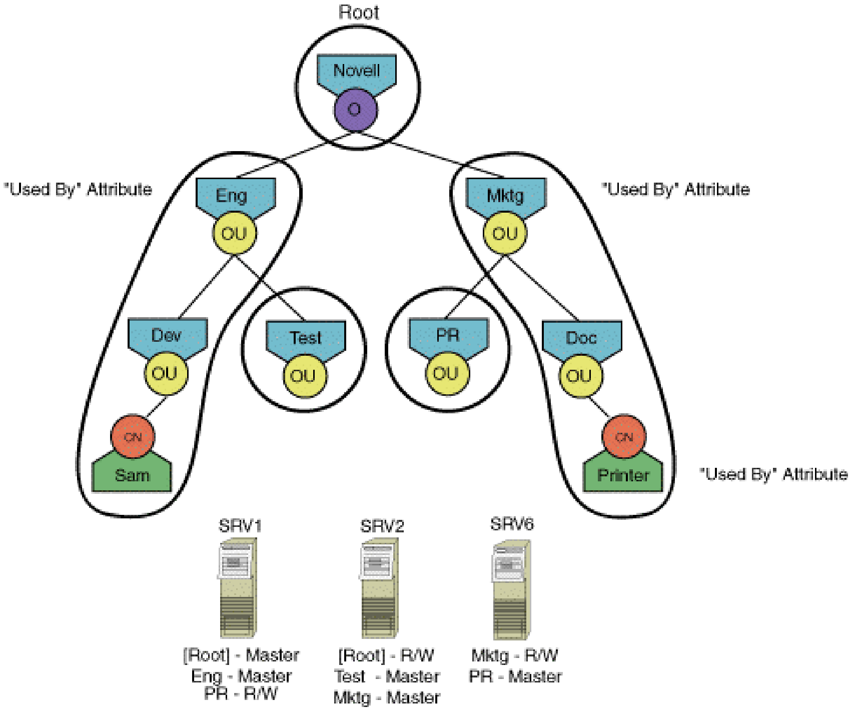
4.3.3 Distributed Reference Links (DRLs)

Distributed Reference Links (DRLs) work with external references to maintain connectivity in the eDirectory tree. They are new in NetWare 5.0 and replace back links. They have the advantage of referencing a partition rather than a specific server. Back links require a server to communicate with every server that contains a read/write replica of the partition the back link resides on. When information is needed about a DRL, any server with a replica of the partition can supply the information.

For backward compatibility with NetWare 4.x servers, eDirectory maintains back links with NetWare 4.x servers. However, these servers must be running at least NDS version 599a in a mixed NetWare 4.x and NetWare 5.x network.

Before an external reference is created in NetWare 5.x, eDirectory places a Used By attribute on a writable copy of the referenced object. This attribute is also placed on the referenced object's partition root and the object's partition root. The following figure illustrates this process.

Figure 4-7 *DRL Process*



In this example, Sam is given rights to use Printer.Doc.Mkgt.Novell. Since this printer resides on another partition, and the SRV1 server does not contain a replica of the Mktg partition, SRV1 needs to create DRLs before creating the external reference to the printer. eDirectory places a Used By attribute on the printer, on the printer's root partition (Mktg), and on Sam's root partition (Eng). Since these are attribute values, any server containing a replica of the partitions can answer queries. In the example above, both SRV2 and SRV6 could answer queries about the referenced printer object.

The Used By attribute on a partition root contains information about objects that are referenced and objects that are referencing other objects. In the example above, the attribute on the Eng partition would indicate that Sam is referencing an object on the Mktg partition. The attribute on the Mktg partition would indicate that an object on the Eng partition has a reference to an object on the Mktg partition.

The attribute on an objects that is referenced contains information about the object that created the external reference. In this example, the Used By attribute on the printer would indicate that Sam is referencing it.

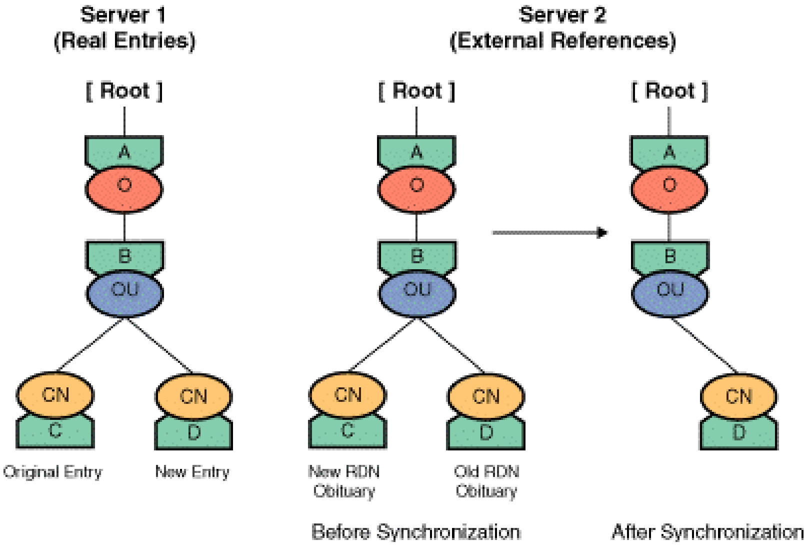
4.3.4 Obituaries

In a distributed database, each server receives updated information through synchronization. Because the servers do not receive updates simultaneously, the servers may not hold the same information at a given time. For this reason, each server holds on to the old information until all the other servers receive updates. eDirectory uses obituaries to keep track of such information.

Obituaries are attribute values that are not visible to clients and are used in server-to-server exchanges.

For example, the figure below shows how obituaries are used when an entry is renamed. On Server 1, the entry C is renamed to D. When Server 2, which holds a replica of C, receives the update during synchronization, it keeps the copy of C and attaches an obituary (called New RDN obituary in the figure). This obituary, which points to the new object, ensures that all servers can access C, even if they have not been notified of the name change. When Server 2 creates entry D, it attaches an Old RDN obituary pointing back to the original object. After all replicas have been synchronized, server 2 can delete its copy of C and remove the obituary from entry D.

Figure 4-8 *Obituaries*



Since obituaries are attribute values, eDirectory synchronizes them the same way it synchronizes other attribute values in the replicas.

Primary and Secondary Obituaries. eDirectory uses two types of obituaries: primary and secondary. Primary obituaries keep track of entry-level modifications, including

- Renaming an entry
- Deleting an entry
- Moving an entry
- Moving a subtree

Generally, when data is changed, primary obituaries convey the changes to servers holding the affected entry.

Secondary obituaries convey the change to servers holding external references to the changed entry. The secondary obituary, also called a back link obituary, is placed on the entry's back link when the entry is renamed, moved, or deleted.

Obituary States. Both primary and secondary obituaries go through several states before they can be deleted from the database. Since secondary obituaries are linked to primary obituaries, all secondary obituaries must reach the state before the primary obituary can be marked as reaching that state.

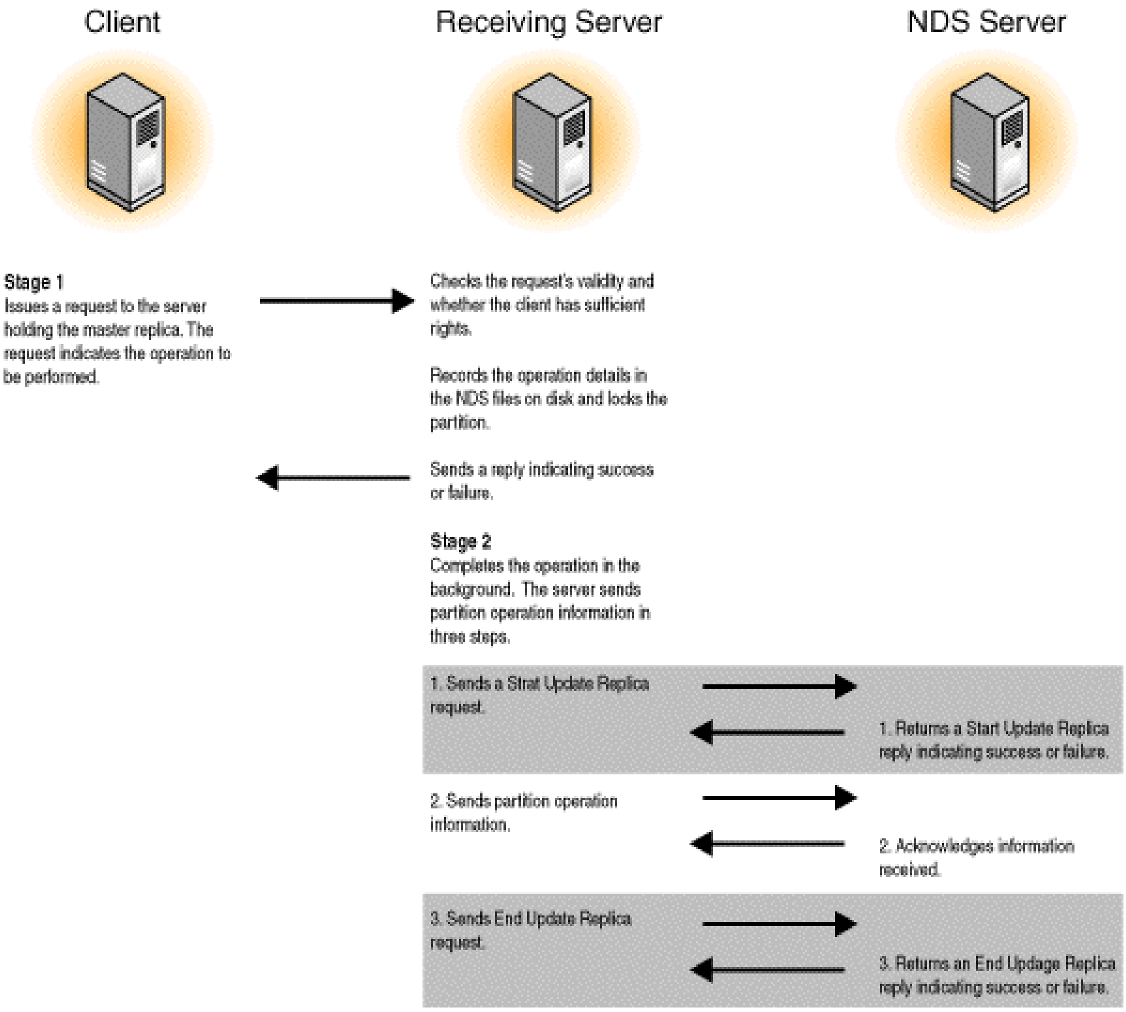
Obituaries progress through the following states:

- Initial state assigned at creation
- Notified servers with copies of the obituary have been notified
- Wait timeout to ensure all changes are propagated to all replicas
- Purgeable servers with copies of the obituary mark the obituary as purgeable
- Wait timeout to ensure all changes are propagated to all replicas. Once they have been propagated, the obituary is deleted from the database.

4.4 Partition Operations

eDirectory allows administrators to create and manage partitions and their replicas. These operations, called partition operations, allow great flexibility in maintaining and modifying an eDirectory tree. This section describes the available operations and the processes behind them. All operations involve two major stages. Stage 1 is the processing of the client request by the Directory Services Agent (DSA) layer and its reply to the client. Stage 2 is the background processing of the request which usually involves requests to other eDirectory servers. The figure below illustrates these stages.

Figure 4-9 *Partition Operations*



Stage 1
Issues a request to the server holding the master replica. The request indicates the operation to be performed.

Checks the request's validity and whether the client has sufficient rights.

Records the operation details in the NDS files on disk and locks the partition.

Sends a reply indicating success or failure.

Stage 2
Completes the operation in the background. The server sends partition operation information in three steps.

1. Sends a Strat Update Replica request.

1. Returns a Start Update Replica reply indicating success or failure.

2. Sends partition operation information.

2. Acknowledges information received.

3. Sends End Update Replica request.

3. Returns an End Updage Replica reply indicating success or failure.

During Stage 2, servers use the replica synchronization protocol to exchange information and move the replicas through the various states associated with the partition operation (see "Replica Transition States" on page 69). The replica synchronization protocol ensures that the other replica is in a state to perform a synchronization and that the information about the partition operation, once received, has been successfully applied. If the operation requires more than one state change, the servers exchange multiple sequences of replica update information.

For partition operations to succeed,

- All servers in the replica ring must be up.
- In NetWare 4.x, all servers must be able to talk to each other.
- In NetWare 5.x, servers with master replicas must be able to talk to each other and to at least one server in their replica ring. The master replica does not need to talk to each and every replica in the ring.

Partition operations include the following:

- "Adding a Replica" on page 70
- "Changing Replica Types" on page 75
- "Removing a Replica" on page 72
- "Joining Two Partitions" on page 77
- "Moving a Subtree" on page 80
- "Splitting a Partition" on page 74

4.4.1 Replica Transition States

A client can add or delete a replica when another partition operation is taking place. The other partition operations (changing, joining, splitting, moving) cannot occur simultaneously but must be queued to occur sequentially. Partition operations use replica states to indicate the status of the partition and its replicas as they transition to a new state.

NetWare 4.x and NetWare 5.x use different transition states for some partition operations. In a mixed eDirectory tree that contains both NetWare 4.x and NetWare 5.x servers, the master replica determines behavior.

If the master replica is on a NetWare 5.x server,

- Nonmaster replicas on NetWare 5.x servers use NetWare 5.x transition states.
- Nonmaster replicas on NetWare 4.x servers use NetWare 4.x transition states.

If the master replica is on a NetWare 4.x server, all replicas use NetWare 4.x transition states.

The following tables lists the partition operations those transitions states varying according to the NetWare operating system version.

Table 4-3 *Partition Operations Versus NetWare Operating System*

Partition Operation	NetWare 4.x Transition States	NetWare 5.x Transition States
Adding a replica	1st state: RS_NEW_REPLICA	1st state: RS_BEGIN_ADD
	2nd state: RS_TRANSITION_ON	2nd state: RS_NEW_REPLICA
	3rd state: RS_ON	3rd state: RS_ON
Removing a replica	1st state: RS_DYING_REPLICA	1st state: RS_DYING_REPLICA
	2nd state: removes itself	2nd state: RS_DEAD_REPLICA
		3rd state: removes itself
Changing which replica is the master	1st state: RS_CRT_0	1st state: RS_MASTER_START
	2nd state: RS_CRT_1	2nd state: RS_MASTER_DONE
	3rd state: RS_ON	3rd state: RS_ON

Partition Operation	NetWare 4.x Transition States	NetWare 5.x Transition States
Moving a subtree to another location	1st state: RS_MS_0 2nd state: RS_ON	1st state: RS_MS_0 2nd state: RS_MS_1 3rd state: RS_ON

Some partition operations in NetWare 4.x and NetWare 5.x use the same transition states. The following table lists these operations and their states.

Partition Operation	Transition States
Splitting a partition into two partitions	1st state: RS_SS_0 2nd state: RS_SS_1 3rd state: RS_ON
Joining two partitions into one partition	1st state: RS_JS_0 2nd state: RS_JS_1 3rd state: RS_JS_2 4th state: RS_ON

4.4.2 Adding a Replica

Adding a replica causes a new replica to be added to a partition’s replica list and places a replica on the specified server. NetWare 4.x and NetWare 5.x both use a two stage process, but the methods are slightly different.

Adding a Replica in NetWare 4.x

In Stage 1, the client sends the request to an eDirectory server with the server's name, the partition name, and the replica type. The request is sent to the server with the master replica.

The master server verifies that

- Target server exists and doesn't currently hold a replica of the partition.
- Client has sufficient rights to the target server and the partition's root object.
- Partition is not currently involved in another partition operation that prevents an add. Another add or remove operation can be in progress, but any other partition operation will cause the request to be denied.

If the request is valid, the master server starts the process by

- Creating subordinate references requests for any partitions that are below the newly added partition (these requests will be sent to the target server).
- Adding the replica to replica list.
- Setting the replica's state to RS_NEW_REPLICA.
- Sending the client reply indicating success or failure.

In Stage 2, the master server communicates the request to the target server which will add the replica. The master server sends the following to the target server:

- Name of the replica's partition root
- All entries for the replica being added
- All updates to the entries
- The replica's state change to RS_TRANSITION_ON when the target server acknowledges receiving all updates
- The replica's state change to RS_ON when the target server acknowledges receiving updates from the other servers in the replica ring

The target server completes the following tasks:

- Contacts the master replica of each subordinate reference to create a subordinate reference for the target server.
- Adds each subordinate reference as the information comes from each master replica.
- Adds the replica.
- Receives the replica's entries and updates from the master server.
- Requests updates from all servers in the replica ring other than the master.
- Requests a change state to RS_ON when updates have been received from the other servers in the replica ring.

Adding a Replica in NetWare 5.x

In Stage 1, the client sends the request to an eDirectory server with the server's name, the partition name, and the replica type. The request is sent to the server with the master replica.

The master server verifies that

- Target server exists and doesn't currently hold a replica of the partition.
- Client has sufficient rights to the target server and the partition's root object.
- Partition is not currently involved in another partition operation that prevents an add. Another add or remove operation can be in progress, but any other partition operation will cause the request to be denied.

If the request is valid, the master server starts the process by

- Contacting the master replica of each subordinate reference to add a subordinate reference to the target server.
- Adding the replica to replica list.
- Setting the replica's state to RS_BEGIN_ADD.
- Sending the client reply indicating success or failure.

In Stage 2, the master server communicates the request to the replica ring. The master server completes the following tasks:

- Synchronizes its replica list with the replica ring.
- Changes the new replica's state to RS_NEW_REPLICA when subordinate references on the target server are in an RS_ON state.

- Changes the replica's state to RS_ON when the Transitive Vector indicates that the target server has seen the RS_NEW_REPLICA state.

The target server completes the following tasks:

- Adds the subordinate references for the partitions below the replica.
- Adds the replica.
- Receives an inbound synchronization that contains the replica's entries and updates.
- Waits until the end of a successful inbound synchronization with one server in the replica ring before synchronizing with other servers in the ring.
- Sends transitive vector updates to the replica ring as part of outbound synchronization.

4.4.3 Removing a Replica

You can remove a replica from a replica list or ring as long as the requester has managed rights to the partition's root entry, managed rights to the target server which holds the replica, and the target replica is not the master replica. If the target replica is the master replica, another replica must be designated as the master replica before the target replica can be removed.

The NetWare 4.x and NetWare 5.x processes are slightly different and use different replica states. However, both use a two stage process, and the first stage is identical for both. The second stage is different and is described separately for each.

Stage 1. In Stage 1, the client request is received by the master server and either accepted or denied.

The client sends the master server a request that contains

- The name of the replica to remove.
- The name of the server that holds the replica.

The master server verifies that

- The client has the rights to remove the replica.
- The server exists and holds the specified replica.
- The partition is not involved in another partition operation that prevents an add. Another add or remove operation can be in progress, but any other partition operation will cause the request to be denied.

If the request is valid, the master server starts the process by

- Changing the replica's state to RS_DYING_REPLICA.
- Sending the client a reply indicating success or failure.

NetWare 4.x. In Stage 2, the master server must contact the target server which holds the replica that is being removed.

The master server completes the following tasks:

- Sets the Replica attribute to dying and synchronizes it to the target server.
- Deletes the replica or converts it to a subordinate reference, depending on the request from the target server.

The target server completes the following tasks:

- Completes a successful inbound synchronization with the master server that contains the RS_DYING_REPLICA state.
- Completes a successful outbound synchronization that contains the state change with another replica that is not in the RS_DYING_REPLICA state.
- Sends the master replica a request to delete the target replica (if the parent replica wasn't local) or to convert the target replica to a subordinate reference (if the parent replica was local).
- Converts the target replica's root entry and the children to either subordinate or external references:
  - If the parent replica is local, converts the replica to a subordinate reference and the children to external references.
  - If the parent replica is not local, converts the replica and its children to external references.

NetWare 5.x. In Stage 2, the master server can communicate the operation to any server in the replica ring, and any server in the replica ring can forward the operation to the target server.

The master server completes the following tasks:

- Sets the Replica attribute to dying and synchronizes the change with the replica ring.
- Changes the replica's state to RS_DEAD_REPLICA when the Transitive Vector indicates that target server has seen the RS_DYING_REPLICA state.
- Deletes the replica or converts it to a subordinate reference, depending on the flag in the Transitive Vector that indicates whether the target server has the parent replica.

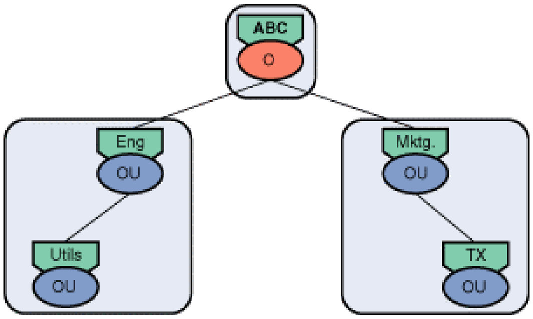
The target server completes the following tasks:

- Completes a successful inbound synchronization that contains the RS_DYING_REPLICA state.
- Completes a successful outbound synchronization that contains the state change with another replica that is not in the RS_DYING_REPLICA state.
- Converts the target replica's root entry and the children to either subordinate or external references:
  - If the parent replica is local, converts the replica to a subordinate reference and the children to external references.
  - If the parent replica is not local, converts the replica and its children to external references.

4.4.4 Splitting a Partition

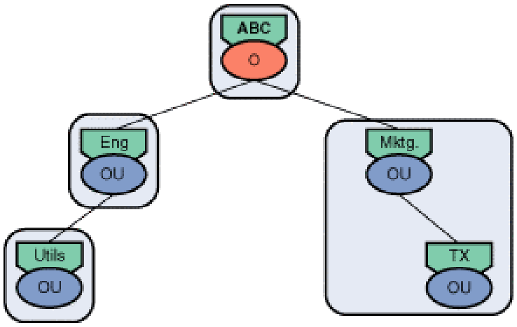
You can split any partition as long as the new partition will have a container entry as its root entry. For example, in the figure below, you might want to create a new partition with Utils.Eng.ABC as the new partition root.

Figure 4-10 *A Partition before a Split*



After Utils has been split into a new partition, the tree structure looks like this.

Figure 4-11 *A Partition after a Split*



The NetWare 4.x and NetWare 5.x processes are slightly different. However, both use a two stage process, and the first stage is identical for both. The second stage is different and is described separately for each.

Stage 1. The client sends a request to split the partition with the name of the child partition root.

The master server verifies that

- The client has the rights to split the partition.
- The partition is not involved in another partition operation.
- All replicas are in an RS_ON state.

If the request is valid, the master server starts the process by

- Changing all replicas of the parent partition to RS_SS_0 state, setting the Partition Control attribute to RS_SS_0 state, and adding the name of the new child partition to the Partition Control attribute.
- Sending the client a reply indicating success or failure.

NetWare 4.x. In Stage 2, the master server must contact every server in the replica ring.

The master server completes the following tasks:

- Processes any obituaries that may be stored on the new partition root.
- Sends a split request to each server in the replica ring.
- Advances the state of each replica to RS_SS_1 as each replica indicates that it has successfully split the partition.
- Advances all replicas to RS_ON when all replicas reach the RS_SS_1 state.
- Sets the Partition Control attribute to idle, indicating that the operation has completed.

The servers in the replica ring complete the following tasks:

- Split the partition when the request is received.
- Return to the master server a reply indicating the successful completion of the split.

NetWare 5.x. In Stage 2, the split partition operation uses the following process. The master server completes the following tasks:

- Processes any obituaries that may be stored on the new partition root.
- Completes a successful outbound synchronization that contains the split information in the Replica and Partition Control attributes.
- Advances the replicas and the Partition Control attribute to the RS_SS_1 state when the master server's Transitive Vector attributes indicate that all replicas have seen the RS_SS_0 state.
- Advances all replicas to RS_ON and the Partition Control attribute to idle when the master server's Transitive Vector attributes indicate that all replicas have seen the RS_SS_1 state.

The servers in the replica ring complete the following tasks:

- Split the partition when they receive the RS_SS_1 state.
- Modify their transitive vector to indicate they received the RS_SS_1 state.
- Continue to perform outbound synchronization during the splitting states.

4.4.5 Changing Replica Types

You can change a nonmaster replica’s type, such as changing a read-only replica to a read/write replica. The process is different when the operation is used to assign a new master replica.

To change a nonmaster replica’s type, a client sends a Change Replica Type request to the server holding the partition’s master replica. The server then changes the replica type in its replica list and propagates the change to the other servers in the replica ring.

The Change Replica Type process can also be used to change where the master replica is stored. For this routine to succeed, the target server must hold a read/write or read-only replica of the partition. In a replica ring with mixed versions of eDirectory, the master replica cannot be changed to a server which is running an older version of NDS. The NetWare 4.x and 5.x processes are slightly different and use different replica states.

Changing the Master Replica in NetWare 4.x

In Stage 1 on NetWare 4.x, NDS uses the following process to change which replica is the master replica. The client sends the Change Replica Type to the source server containing the master replica.

The source server completes the following tasks:

- Verifies that the operation can take place by checking the client rights and ensuring that the target server contains a replica of the partition.
- Sets the state of the Partition Control attribute to RS_CRT_0 and stores the name of the new master server in the attribute.
- Sets the state of all replicas in the ring to RS_CRT_0.
- Returns a Change Replica Type reply to the client.

In Stage 2, the source server completes the following tasks:

- Synchronizes the state change to all the servers in the replica ring.
- Sends the target server a verb to change its replica type to master.
- Changes its replica type to read/write.

The target server, which will contain the new master replica, completes the following tasks:

- Changes its replica type to master.
- Sets all replicas in the ring to RS_CRT_1.
- Synchronizes the state change to all servers in the replica ring.
- When the synchronization has successfully completed, changes all replicas to RS_ON and sets the Partition Control to the idle state.

Changing the Master Replica in NetWare 5.x

In Stage 1 on NetWare 5.x, NDS uses the following process to change which replica is the master replica. The client sends the Change Replica Type to the source server containing the master replica.

The source server completes the following tasks:

- Verifies that the operation can take place by checking the client rights and ensuring that the target server contains a replica of the partition.
- Sets the state of the old master replica and the new master replica in the ring to RS_MASTER_START.
- Returns a Change Replica Type reply to the client

In stage 2, the source server completes the following tasks:

- Synchronizes the state change to the replica ring.
- When its Transitive Vector attributes indicate that the target server with the new master has received the state change, sets the state of old master replica and the new master replica to RS_MASTER_DONE.
- Synchronizes the state change to the replica ring.
- When its Transitive Vector attributes indicate that the target server has accepted the state change, changes its replica type to Read/Write.

The target server, which will contain the new master replica, completes the following tasks:

- When it receives the RS_MASTER_DONE state, changes its replica type to Master.

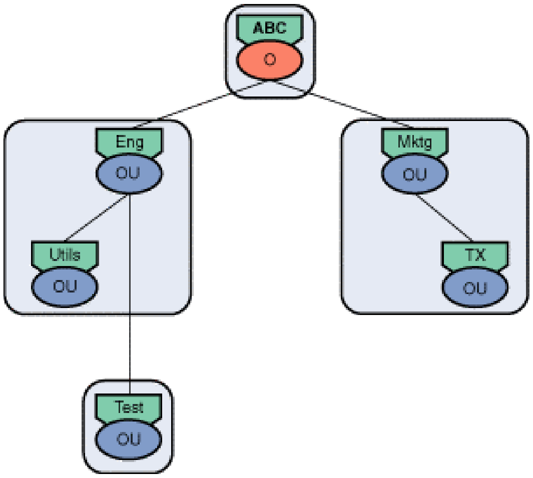
- When its Transitive Vector attributes indicate that all servers in the replica ring have received the RS_MASTER_DONE state, sets the state of all replicas in the ring to RS_ON and changes the Partition Control state to idle.

4.4.6 Joining Two Partitions

You can join a partition with its parent partition. Replicas of the two partitions do not have to be stored on the same servers. The Join Partitions operation makes the Replica list the same for both the parent and child partition, excluding the subordinate references from the parent's replica list. When eDirectory creates all the replicas of the new partition, it erases the partition boundary.

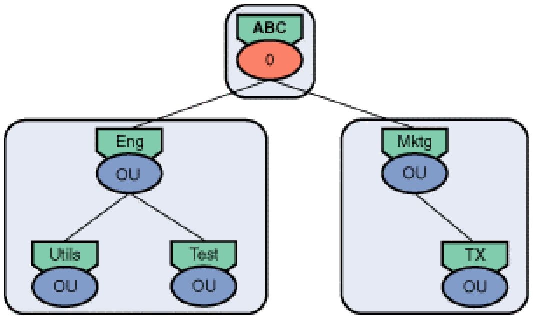
For example, you might want to join the Test partition with its parent partition, Eng.

Figure 4-12 *A Partition before a Join*



After the operation, the partitions would look like this.

Figure 4-13 *A Partition after a Join*



NetWare 4.x and NetWare 5.x uses the same procedures in Stage 1 and slightly different procedures in Stage 2.

Joining Operation in NetWare 4.x

Stage 1. The client sends the master replica of the child partition a Join Partitions request.

The server with the child master replica performs the following tasks:

- Validates the request by verifying (1) that every server in the ring is running an NDS version equal to or greater than 435, (2) that the client has sufficient rights for the request, and (3) that the child partition is not involved in another partition operation.
- Sets the state in the Partition Control attribute to RS_JS_0 and stores the current operation (PC_JOINING_UP) in the attribute.
- Sets all the replicas of the child partition to the RS_JS_0 state.
- Sends a Join Start request to the server holding the master replica of the parent partition.

The server with the parent master replica performs the following tasks:

- When it receives the Join Start request, sets the state of the Partition Control attribute to RS_JS_0 and stores the name of the child partition and the current operation (PC_JOINING_DOWN) in the attribute.
- Sets all the replicas of the parent partition to the RS_JS_0 state.
- Sends the server with the child master replica a Join Start reply.

When the server with the child master replica receives the Join Start reply from the parent master, the child master sends the client a Join Partitions reply.

Stage 2. Both servers propagate changes to the servers in their replica rings. The server with the parent master replica performs the following tasks:

- Synchronizes the state change to all replicas.
- Obtains the servers in the child's replica ring.
- Sends an Add Replica requests to add parent replicas to the servers that have the child replica but not the parent. The parent replicas are set to the same type as the child (read/write or read-only).
- Waits while the replicas are added.
- When all the new replicas achieve an RS_ON state, sets all replicas in the ring to an RS_JS_0 state.
- Sets the state in the Partition Control attribute to RS_JS_1.
- Waits for the state of the child's Partition Control attribute to achieve RS_JS_2.

The server holding the child master replica performs the following tasks:

- Synchronizes the state change to all replicas.
- Obtains the servers in the parent's replica ring.
- Sends an Add Replica request to add child replicas to the servers that have the parent replica but not the child. The child replicas are set to the same type as the parent (read/write or read-only).
- Waits while the replicas are added.
- When all the new replicas achieve an RS_ON state, sets all replicas in the ring to an RS_JS_0 state.
- Sets the state in the Partition Control attribute to RS_JS_1.
- Waits for the state of the parent's Partition Control attribute to achieve RS_JS_1.

- When the parent's attribute achieves RS_JS_1, sets its attribute to RS_JS_2 and the replicas in the ring to RS_JS_2. (The replicas are never set to RS_JS_1.)

The server holding the parent master replica checks the state of the Partition Control attribute of child partition. When it reaches RS_JS_2, the server performs the following tasks:

- Sends a low level join request to all servers holding replicas of the two partitions. As each server receives the request, the server erases the boundaries between the partitions.
- Performs a low level join on its parent and child partition.
- Sets the replica state of the parent partition to RS_JS_2.
- Synchronizes the changes to all replicas.
- When the synchronization is completed, sets all replica states to RS_ON.

Joining Operation in NetWare 5.x

Stage 1. The client sends the master replica of the child partition a Join Partitions request.

The server with the child master replica performs the following tasks:

- Validates the request by verifying (1) that every server in the ring is running an NDS version equal to or greater than 435, (2) that the client has sufficient rights for the request, and (3) that the child partition is not involved in another partition operation.
- Sets the state in the Partition Control attribute to RS_JS_0 and stores the current operation (PC_JOINING_UP) in the attribute.
- Sets all the replicas of the child partition to the RS_JS_0 state.
- Sends a Join Start request to the server holding the master replica of the parent partition.

The server with the parent master replica performs the following tasks:

- When it receives the Join Start request, sets the state of the Partition Control attribute to RS_JS_0 and stores the name of the child partition and the current operation (PC_JOINING_DOWN) in the attribute.
- Sets all the replicas of the parent partition to the RS_JS_0 state.
- Sends the server with the child master replica a Join Start reply.

When the server with the child master replica receives the Join Start reply from the parent master, the child master sends the client a Join Partitions reply.

Stage 2. Both servers propagate changes to the servers in their replica rings.

The server holding the child master replica performs the following tasks:

- Obtains the servers in the parent's replica ring.
- Sends an Add Replica request to add child replicas to the servers that have the parent replica but not the child. The child replicas are set to the same type as the parent (read/write or read-only).
- Waits while the replicas are added.
- When all the new replicas achieve an RS_ON state, sets all replicas in the ring to an RS_JS_0 state.

- When all replicas have seen the RS_JS_0 state, sets the state in the Partition Control attribute and all replicas to RS_JS_1.
- When all replicas have seen RS_JS_1 or RS_JS_0, sets the Partition Control attribute and all replicas to RS_JS_2.

When the child replicas have reached RS_JS_2, the server with the child master replica has completed its tasks. The server with the parent master replica now treats it as just another server with copies of both the parent and child partitions.

The server with the parent master replica performs the following tasks:

- Obtains the servers in the child's replica ring.
- Sends an Add Replica request to add parent replicas to the servers that have the child replica but not the parent. The parent replicas are set to the same type as the child (read/write or read-only).
- Waits while the replicas are added.
- When all the new replicas achieve an RS_ON state, sets all replicas in the ring to an RS_JS_0 state.
- Sets the state in the Partition Control attribute to RS_JS_1 and synchronizes the changes.
- Waits until all replicas have seen RS_JS_1, then sets the replicas to RS_JS_2 and synchronizes the changes.
- When each replica receives the RS_JS_2 state, it erases the boundaries between the partitions. When all replicas have seen RS_JS_2, the server with the parent master replica erases the boundaries between the partitions.
- Sets the Partition Control attribute to idle and all replica states to RS_ON.

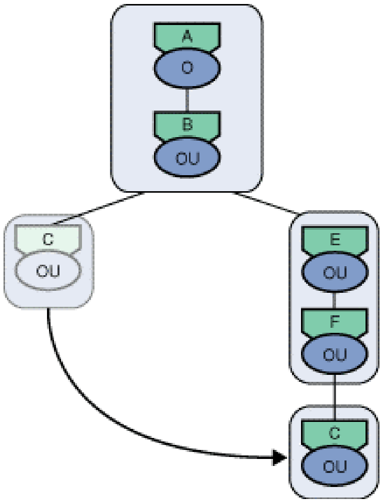
4.4.7 Moving a Subtree

eDirectory allows you to move any subtree under any container entry that meets the schema rules as long as the subtree to be moved has no child partitions.

The Move Subtree operation moves a container with all its child entries as a unit to another logical location in the eDirectory tree. No entries join or leave the partition during this operation, and the replicas of the partition are held on the same servers they were held on previously. In other words, eDirectory moves the subtree logically, not physically. The entries' Distinguished Names change, but their Relative Distinguished Names do not.

For example, you may want to move object C and its subordinate objects under container F. Object C's distinguished name would change from C.B.A to C.F.E.B.A. The subtree that you move must be the partition root The destination does not need to be a partition root, but can be any container within the destination partition.

Figure 4-14 *The Subtree Move Operation*



The Move Subtree operation is a three stage operation. The client makes two requests: one to the server containing the partition to move and the other to the server containing the new parent partition.

State 1. The client sends a Begin Move Entry request to the destination server which holds the master replica of the parent partition to which the child partition is moving. The request identifies the child partition that is moving and its new parent.

The destination server completes the following tasks:

- Verifies that the client has Create [Entry] rights in the destination container.
- Verifies that no other partition operations are in progress (all replicas are on and that the Partition Control attribute is set to idle).
- Sends the client a Begin Move Entry reply.
- Adds the child to its expectation list and sets a 10 minute timer. If the server does not receive a request to start the move operation within 10 minutes, the server clears the child from the expectation list.

Stage 2. The client sends a Finish Move Entry request to the source server. The source server is the server that holds the master replica of the child partition that is moving.

The source server completes the following tasks:

- Verifies the validity of the request by verifying that the partition is not busy with another partition operation and that client has delete rights to the child partition object.
- Receives the name of the server holding the master replicas of the destination container and sends this destination server a Start Move request.

The destination server completes the following tasks:

- Checks the Start Move request against its expectations list.
- Checks that all replicas are on and the Partition Control attribute is set to idle.

- Verifies that all servers in the replica ring are running a version of eDirectory that supports the move operation.
- On the destination's partition root, sets the function field in the Partition Control attribute to PC_MOVE_SUBTREE_DEST and the state field to RS_MS_0.
- Sets each child replica to RS_MS_0.
- Sends the source server a Start Move reply.
- Synchronizes the changes.

The source server completes the following tasks:

- Creates three Partition Control attributes and sets each function field to PC_MOVE_SUBTREE_SRC and each type field to a different type (PCT_0, PCT_1, or PCT_2).
- Sets all parent replicas to RS_MS_0.
- Adds an obituary (OBT_MOVE_TREE) for each server in the replica ring for the source and for each server in the backlink list.
- Sends a Finish Move Entry reply to the client
- Synchronizes the changes.

Stage 3. The source and destination servers complete the move. The client is no longer involved in the process.

4.5 Indexes

eDirectory 8.5 introduced a new feature to improve searching performance, called indexes. Indexes provide a way to create a group of entries based on something they have in common, such as an attribute or an attribute value, and use this group to quickly return information when a search is performed.

Several pre-defined indexes are included with eDirectory, and you can define your own either programatically or using ConsoleOne. eDirectory also provides a mechanism called Predicate Status to view the most often used filters to help you determine which indexes to create.

4.5.1 Index Composition

Indexes are basically composed of two things:

- Attribute to be indexed
- Index matching rule

The first component of an index is the indexed attribute. The second component of an index is the index matching rule, which defines the types of searches that use that attribute.

There are three types of index matching rules:

- **Presence.** This rule tests for the presence of an attribute, such as (cn=*). If a value exists for this attribute then the entry is added to the index.
- **Value.** This rule tests for an exact attribute value, such as (o=acme). Any entry with this value for the specified attribute is added to the index.

- **Substring.** This rule test for a substring in an attribute value, such as (emailaddress=*@acme.com). Any entry with the specified attribute matching this substring is added to the index. These indexes are most costly to maintain but provide the most flexibility.

  To determine which indexes are most useful in your environment, see "Predicate Status" on page 83.

4.5.2 Predicate Status

Predicate Statistics is an eDirectory object that gathers information about search filters to help determine the correct indexes to use. This mechanism is turned off by default and can be enabled on the NDS Server object using ConsoleOne. Because of the overhead, this mechanism should only be used to gather data then disabled.

Index Overhead

When an entry is added, deleted or modified in eDirectory, all all corresponding indexes must also be updated. Because of this overhead.

Because there is overhead involved whenever an object associated with an index is added, deleted, or modified, it is important to carefully determine which indexes to use.

4.6 Synchronization

Synchronization is the process of ensuring that all changes to a particular partition are made to every replica of that partition. The X.500 standard defines two synchronization mechanisms:

- Master-to-slave
- Peer-to-peer

The master-to-slave mechanism requires that all changes be made on the master replica. That replica is then responsible to update all the other replicas (slave replicas).

In a peer-to-peer synchronization system, updates can be made to any read-write or master replica. At a predetermined interval, all servers holding copies of the same partition communicate with each other to determine who holds the latest information for each object. The servers update their replicas with the latest information for each replica.

eDirectory uses both the master-to-slave and peer-to-peer synchronization processes, depending upon the type of change being made. Most operations use peer-to-peer synchronization. The biggest exceptions are the partition operations which require a single point of control. In NetWare 4.x, the server with the master partition communicates with all other servers in the replica ring and controls the updates for partition operations. In NetWare 5.x, the master-to-slave mechanism has been slightly modified for partition operations. The server with the master replicas isn't responsible for updating all replicas. This server is responsible for updating its own replica and one other replica. The other replicas in the ring are updated peer-to-peer.

In eDirectory, the synchronization time interval ranges from between 10 seconds to 25 hours depending upon the type of information updated.

To understand the eDirectory synchronization processes, you should be familiar with the following eDirectory features:

- "Loose Consistency" on page 84

- "Time Stamps" on page 84
- "Partition Information" on page 85

eDirectory has a number of synchronization processes. From a user's view point, the most important one is Replica Synchronization (page 85) which keeps the object information synchronized across all replicas. The others are Background Processes (page 88) that update eDirectory information that is mostly hidden from the user but required to ensure the efficient distribution and replication of eDirectory.

4.6.1 Loose Consistency

Because the eDirectory database must synchronize replicas, not all replicas hold the latest changes at any given time. This concept is referred to as loose consistency (called transient consistency in the X.500 standard), which simply means that the partition replicas are not instantaneously updated. In other words, as long as the database is being updated, the eDirectory database is not guaranteed to be completely synchronized at any instance in time. However, during periods in which the database is not updated, it will completely synchronize.

Loose consistency has the advantage of allowing eDirectory servers to be connected to the network with different types of media. For example, you could connect one portion of your company's network to another by using a satellite link. Data traveling over a satellite link experiences transmission delays, so any update to the database on one side of the satellite link is delayed in reaching the database on the other side of the satellite link. However, because the database is loosely consistent, these transmission delays do not interfere with the normal operation of the network. The new information arrives over the satellite link and is propagated through the network at the next synchronization interval.

Another advantage to loose consistency is that if part of the network is down, the changes will synchronize to available servers. When the problem is resolved, the replicas on the affected servers will receive updates.

eDirectory is a loosely synchronized database. Because of the time required to synchronize different servers, eDirectory does not guarantee that a particular piece of information on a particular replica has the most current value. If a client agent were to request the same information from two replicas, there is a chance that the results may differ. This happens when an update is made in one replica and the client agent reads from another replica before the update is synchronized.

Loose synchronization allows eDirectory to reduce the overhead required to maintain a partition. eDirectory is designed to support frequent reads and infrequent writes. In light of this, clients are discouraged from storing values in eDirectory that change frequently.

4.6.2 Time Stamps

One critical component in synchronization is the time stamp, which records information about when and where a given value in a given attribute was modified. When eDirectory updates a replica, it sends a modification time stamp with the data to be updated. The replica compares time stamps and replaces the old information with the new.

4.6.3 Partition Information

For normal operations, including synchronization, to be successful, the partition root object on each server must store several important attributes and their values:

- **Replica Pointers.** The local server stores a pointer to the remote servers that contain a replica of this partition. This pointer structure contains, among other things, the server's ID, address, and the type of replica stored on the server.
- **Partition Control Attribute.** The local server uses this attribute to track the progress of operations such as splitting and joining partitions and repairing time stamps.
- **Partition Status Attribute.** The local server uses this attribute to store information about the success of the last synchronization cycle.
- **Synchronized Up To Attribute.** NetWare 4.x servers use this attribute to store a time stamp that indicates the last update the local replica received from each remote replica.
- **Transitive Vector Attribute.** NetWare 5.x servers use this attribute to store information about each replica in the ring, and the attribute includes a modification time stamp.

4.6.4 Replica Synchronization

Replica synchronization refers to the process of copying data among the replicas of a partition. A partition is synchronized if all of its replicas contain the same information. If one replica has a more current version of a piece of data than the other replicas, it propagates this data to the other replicas. The act of performing synchronization of the replicas of a partition is called the convergence of the replicas. The time between scheduled synchronization is called the synchronization interval.

All replicas belonging to a given partition set must be synchronized to include all updates generated to writable copies of the partition set (master and secondaries). This task is the responsibility of the replica synchronization process.

When a name server receives an update, it schedules the replica synchronization process to begin in 10 seconds. If the synchronization fails, then the replica synchronization process reschedules the event.

The replica synchronization process is responsible for the following maintenance operations:

- Accepting updates from other replica synchronization processes and applying them to the relevant local replicas.
- Maintaining consistency of the eDirectory tree by performing periodic checks on the links in the tree.
- Maintaining the validity of external references found on the local server.
- Summarizing access control information for each local partition.

The consistency or convergence of the eDirectory replicas is directly related to the frequency with which the replica synchronization processes operate. Administrators can control the frequency of the replica synchronization according to the needs of their system by adjusting the synchronization interval.

Because an eDirectory partition can be replicated and distributed across a network, any changes made to one replica must be sent to, or synchronized with, the other replicas.

eDirectory uses two timers to synchronize changes to replicas:

- Convergence attribute-some changes, such as a change in a user's password or access rights, need to be sent immediately to another replica. When these high convergence attributes are modified, the replica synchronization process is scheduled with the fast synchronization interval of 10 seconds.
- Heartbeat-less critical changes, such as a user's last login time, can be collected locally for a short period of time before being sent to other replicas. The heartbeat triggers a scheduled synchronization at least every thirty minutes. The network administrator can adjust the trigger's time interval with the use of the DSTrace console SET command. eDirectory adjusts the time if it must respond to expiring time outs.

eDirectory uses both a send and receive sync process:

- In bound sync process—receives updates from other replicas. DSTrace commands can turn this process off for a specified interval.
- Out bound sync process—discovers which replicas need updates and then sends the updates.DSTrace commands can turn this process off for a specified interval.

The out bound synchronization process checks the synchronization status of every server that has a replica of a given partition. Factors that determine whether synchronization is necessary include the replica's convergence attribute, its replica type, and the time that has elapsed since the replica was last synchronized or updated. The synchronization process in NetWare 4.x is slightly different than the transitive synchronization process used in NetWare 5.x.

4.6.5 NetWare 4.x Synchronization Process

In NetWare 4.x, the replica synchronization contacts each of the replicas of the partition one by one. After the processing for a replica has been completed, or if it fails, the replica synchronization process tries to process the next replica in the replica list until attempts to update all replicas have been made. It is frequently possible that the replica synchronization process may not be able to update all the replicas in one attempt. If the server holding a replica is currently unavailable, an attempt to update this replica fails. The replica synchronization process, however, proceeds with attempts to update other replicas in the replica list. The replicas that are not updated in this replica synchronization process are attempted again.

When the replica synchronization process begins, it builds a list of replicas for the partition it is attempting to synchronize. The replica synchronization process then sends a StartUpdateReplica message to the server holding the replica that needs to be synchronized.

The receiving server responds with a vector of time stamps that indicate how current the replica is in relation to this partition. The replica synchronization process compares the local Received Up To attribute with the time stamp vector received. If the local Received Up To is more current than the received time stamp vector, the replica synchronization process sends the relevant changes to the receiver. The changes are sent in an UpdateReplica message.

After all the updates have been sent, the replica synchronization process sends an EndUpdateReplica message to the receiver, indicating that all the changes have been sent. If no changes need to be sent (which would be the case if the receiver is as current or more current than the local partition), the replica synchronization process sends an EndUpdateReplica indicating that no changes need to be sent. If the receiver fails to apply the changes, the synchronization is aborted for this replica. The replica synchronization process attempts to synchronize this replica at the next scheduled synchronization.

Once the processing of this replica has been completed, or it has failed, the replica synchronization process begins processing the next replica in the replica list. This procedure is repeated until all the replicas of the partition have been processed.

In NetWare 5.x, NDS uses transitive synchronization instead of replica synchronization if all the replicas in the replica list are stored on NetWare 5.x servers. If the replica list contains a mix of NetWare 4.x and NetWare 5.x servers, the replicas stored on NetWare 4.x servers are synchronized with the replica synchronization process.

4.6.6 Transitive Synchronization

In NetWare 4.x, the replica synchronization process requires that all servers in a replica list be able to communicate and synchronize with each server in the replica list. Each server must be able to query all the servers in the replica list, obtain their synchronization status, and send updates if the server is not synchronized. Transitive synchronization changes this procedure.

Transitive Vector Attribute. Transitive synchronization depends on the Transitive Vector attribute, which is stored as an attribute of the partition root object. The Transitive Vector is a multivalued attribute that stores the following information for each server in the replica ring:

- The server's Distinguished Name (or Entry ID on the local server)
- A count of the replicas in the replica ring
- The last modification time stamp received for each replica in the replica ring
- A replica number for each replica in the replica ring

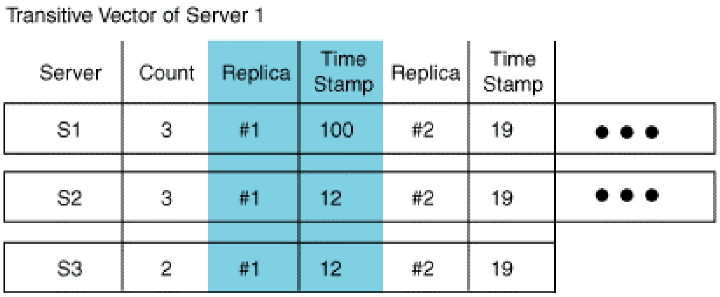
If the server stores two replicas (C and D) and Replica C has three servers in its replica ring and Replica D has two servers in its replica ring, the server will contain five Transitive Vector attributes, one for each replica. Each attribute will contain a value for each server in the ring; thus the attribute for Replica C has three values and the attribute for Replica D has two values.

Starting the Process. When a partition replica is modified on a given server, that server updates the modification time stamp for that replica in its local transitive vector. Then, the server checks its local Transitive Vector for each server in the replica ring. If any server in the Transitive Vector holds a time stamp older than the local time stamp, the local server initiates the synchronization process. Synchronization occurs only between servers that can communicate with each other.

For example, in the figure below, server S1's copy of Replica #1 is modified and given a modification time stamp of 100. Server S1 then checks the local Transitive Vector for servers S2 and S3. Both of these hold time stamps of 12 for Replica #1, indicating that they must be synchronized. Assuming that server S1 can communicate with servers S2 and S3, server S1 initiates the synchronization process. If, however, server S1 is an IPX-only server and S3 is an IP-only server,

server S1 would not initiate replica synchronization with server S3. Rather, server S3 would have to be synchronized with a server capable of using the IP protocol.

Figure 4-15 *Modification Time Stamps*



Transitive Vector of Server 1

Server	Count	Replica	Time Stamp	Replica	Time Stamp	
S1	3	#1	100	#2	19	• • •
S2	3	#1	12	#2	19	• • •
S3	2	#1	12	#2	19	

This procedure reduces synchronization traffic because a server does not have to query each server in the replica ring before synchronizing but checks the local Transitive Vector.

In addition, the procedure allows changes made on one replica to be synchronized to other replicas through intermediaries. This is a requirement with the introduction of IP support. For example, a server that supports only IP cannot communicate directly with a server that supports only IPX. The only requirement in a mixed IP and IPX replica list is that one server in the replica list must support both IP and IPX.

4.7 Background Processes

eDirectory data is loosely consistent, meaning that after a data modification and before synchronization, the data will be different from one replica to another. Several background processes are implemented to ensure synchronization of Directory data over time. The eDirectory background processes run periodically and are responsible for a number of operations that keep the database on all servers in the eDirectory tree synchronized. All of these background processes happen automatically in the background without user intervention; however, some processes allow limited intervention.

eDirectory background processes include

- "Replica Synchronization" on page 85
- "Limber Process" on page 88
- "Back Link Process" on page 89
- "Janitor Process" on page 90
- "Flat Cleaner Process" on page 91

4.7.1 Limber Process

This process maintains tree connectivity. At certain times, each server in the eDirectory tree checks its tree name and modifies it if necessary. When each server in the tree comes up or eDirectory restarts, the Limber process initiates. Also, when a server receives a request to check the tree name from another server in the tree, or for any other reason needs to establish communications with another server in the tree, the Limber process initiates. A request to check the tree name can originate for two reasons.

- When the tree name has changed at the server holding the master replica of the root partition, that server sends the request to all servers in the tree. If the request fails to reach a server, the name will eventually be changed anyway.
- Whenever a server connects to another for server-to-server exchanges, it uses the Ping operation to test the validity of the other server. If it finds that the other server has a different tree name than itself, it sends a request to check the tree name to that server and does not use the connection for server-to-server exchanges.

When the Limber process is initiated, it checks the server's entry to make sure that it is in the right tree, that is, has the correct tree name and address. The Limber process can be initiated sooner than normal by executing a DSTRACE SET console command (SET DSTRACE = *L).

The algorithm for the Limber process proceeds as follows:

When the Limber process is initiated on a server, it first makes a list of server addresses to check that the server is in the right tree. This list includes servers holding replicas of the partition that holds the server object. If the procedure was prompted by a request to check the tree name, the operation inserts the server identified in the request at the head of the list.

The server then sends a Ping request to each server in the list, in order. The Limber process selects the responding server whose root-most entry is closest to the tree root. If two or more servers report the same distance from the root, the Limber process selects the one (if any) whose root-most partition replica is a master replica. Ties are broken by the order of servers in the list.\

If the current server holds the master replica of the root partition and the selected server does not, the process halts.

If the selected server has the same tree name as the current server, the process halts.

Authentication to the selected server is initiated through background authentication procedures. If authentication to that server succeeds, the Limber process assumes that server must be in the same Directory tree. The requesting server then changes its tree name to the one reported by the selected server.

4.7.2 Back Link Process

A back link is stored as an object attribute to keep track of external references to the object. The Directory uses the Back Link attribute to keep track of servers holding external references of an entry. The Back Link attribute has two parts:

- The Distinguished Name(s) of the server(s) holding the external reference; this name is commonly referred to as the Remote Server Name.
- The Entry ID of the remote server, usually referred to as the Remote ID.

When creating an external reference, eDirectory also schedules the creation of a Back Link attribute for the entry. Periodically, the Back Link process checks the external reference to see if the original entry still exists and if there is a reason for the external reference to continue to exist (used in the connection table, file system, or referenced by a local entry). If the external reference is not needed, eDirectory removes it.

The Back Link process enables easy maintenance of the external references by periodically verifying the remote server name and remote ID of each Back Link attribute of entries. The Back Link process checks consistency automatically at intervals of 1500 minutes, or 25 hours. This default back link interval can be changed using a settable parameter in the SET DSTRACE console command (SET DSTRACE = !B [2-10,080 minutes]). Also, the Back Link process flag can be set in the SET DSTRACE console command (SET DSTRACE = *B), which forces the Back Link process to initiate sooner than the next regularly scheduled interval.

When an entry is deleted, back links make it possible for all references to the entry to be deleted. Back links also facilitate renaming and moving entries, because the corresponding changes can be made to the external references through the operation of the Back Link process. Thus, the Back Link process helps to maintain the integrity of external references by allowing them to be updated to reflect the changes made to the objects they refer to. The Back Link process resolves external references to make sure there is a real entry that it refers to, and for real entries the process makes sure that an external reference exists. A local bit in each external reference is used to keep track of the status of back links.

When a server creates an external reference to an entry, it sends a Create Back Link request to a server holding a writable copy of the entry. If the Create Back Link request fails, it retries periodically until the Back Link attribute is created.

When a server removes an external reference from its local database, it sends a Remove Back Link request to a server holding a writable entry for the entry. The server receiving the request checks the Entry Creation Time against the creation time of the actual entry to verify that the request is operating on the correct entry. This Remove Back Link request operation causes the back link to be deleted.

See Also:

- "Back Links" on page 63

4.7.3 Janitor Process

The Janitor process performs two tasks:

- Purges the deleted entries and values that have been synchronized with all the replicas
- Synchronizes external references

The Janitor scans the partition and checks the Replica Up To attribute to determine which values and entries can be purged. A value can be purged from an entry if:

- The value is not present.
- The value's modification time is less recent than the purge time (purge time is the same as the Replica Up To attribute).

The Janitor purges the value if both conditions are met. As part of purging the value, the Janitor performs special processing for counter syntax values to compute the total counter values.

Once the Janitor has inspected all values for an entry, it determines whether the entire entry can be purged. It purges any entry that:

- Is not present.
- Has no values.

While scanning the partition, the Janitor builds two lists:

- Release ID List. This is a list of entries that have been moved successfully. The Janitor releases these entries.
- Notify External Reference list. This is a list of entries that have Back Link obituaries to be processed. The Janitor uses this list to send synchronization messages to external references.

The Janitor also checks if the partition's root entry has been renamed; if so, the Janitor notifies all external references of the root entry of the new name. In addition, the Janitor purges Move expectations that are more than ten minutes old.

The Janitor then uses the two lists it generated in scanning the partitions to synchronize the external references.

For each entry in the Release ID List, the Janitor sends a Release Moved Entry request to the destination entry. If this request is successful, the obituary can then be purged.

For each entry in the Notify External Reference list, the Janitor requests external reference synchronization and specifies the operation that must be performed on the external reference.

4.7.4 Flat Cleaner Process

This process has the following functions:

- For real objects the Flat Cleaner truncates revision count attributes, certifies keys, and generates Certificate Authority keys.
- For Bindery objects and external references, the Flat Cleaner returns deleted space for use.
- For master replicas of NCP server objects, the Flat Cleaner updates the Status and Version attributes.

Bindery Services

5

In versions of NetWare® before version 4.0, information was not available from a distributed directory. Instead, each server in the network stored a database that contained information such as the name, object ID, and password of every user or object that had access to the services provided by that server. Because the servers in the network did not share or communicate this information, the user or object's information was stored separately on every server to which it had rights. For example, if user BrianW had rights to servers Engineering, Mktg, and Sales, his information was stored on each of the three servers. Each time BrianW wanted to access services on a different server, he would have to log in and establish a connection with that server.

Unlike the Directory tree, which is hierarchical, the bindery is a flat structure that is specific to one server. To provide access for bindery users and clients, NDS imitates a flat structure for leaf entries within one or more container entries.

NDS™ emulates the Bindery found in earlier versions of NetWare® by formulating NDS information into a bindery format that can be used to respond to bindery-style requests. This is necessary for backward compatibility with applications that perform bindery functions.

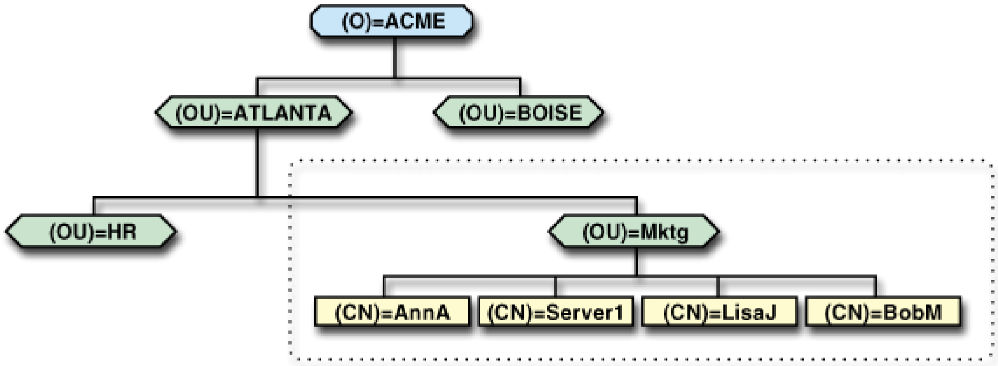
5.1 NDS Bindery Context

The bindery context is the name of the container object(s) where Bindery Services is set. For example, in the figure below, the bindery context is set to Mktg.Atlanta.ACME. You can set the bindery context using a SET command at the console.

Bindery-based clients and servers and NDS entries can access all the entries within containers where a bindery context is set. Bindery services applies only to specific leaf entries within containers where the bindery context is set.

With bindery services, a user logs in to a specific server, which contains a pointer to the containers where the bindery context is set. For example, in the figure below, the bindery context is set to OU=Mktg, which is the container where bindery services is set. The entries within the bindery context are outlined with a dotted line.

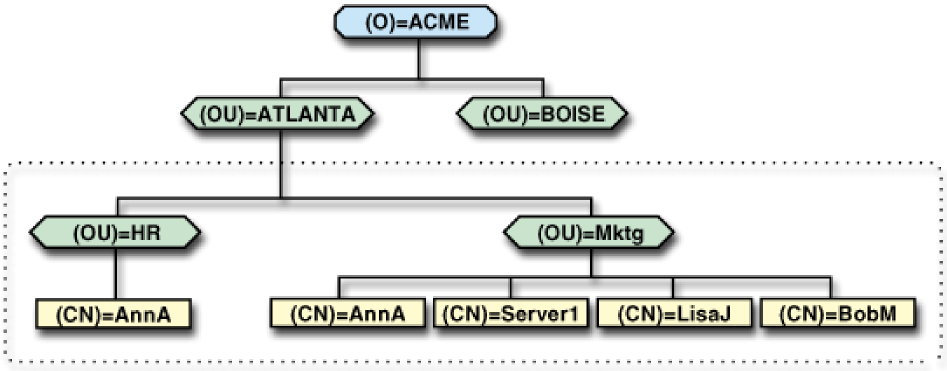
Figure 5-1 *Bindery Services with a Mkgt Bindery Context*



NetWare 2.x and 3.x users see the object in the bindery context as if they were in a real bindery. These entries and dynamic bindery entries are the only entries in the Directory that are available to bindery users. Any server that uses the container as a bindery context must store a read/write replica of the bindery partition.

You can set up to sixteen containers as the bindery context. For example, in the figure below, the bindery context could include OU=HR as well as OU=Mktg.

Figure 5-2 *Two Bindery Contexts*



In this case, NDS uses a bindery context path, which is a list of the containers in the bindery context. For example, the bindery context path for a bindery context containing both OU=HR and OU=Mktg would look like this:

HR.Atlanta.ACME

Mktg.Atlanta.ACME

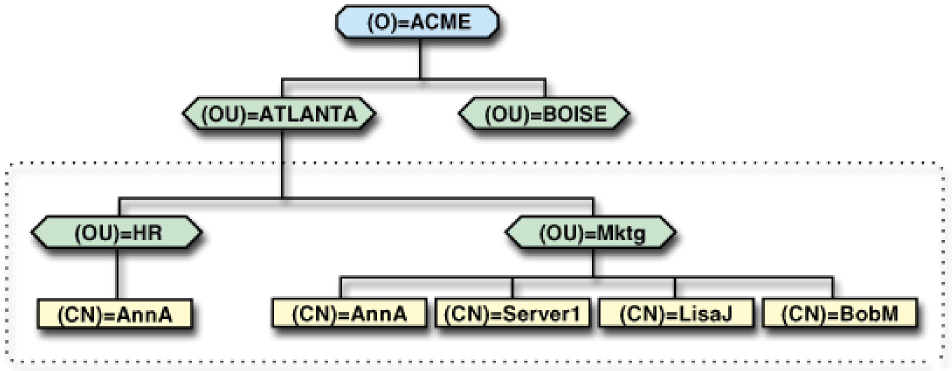
5.2 Bindery Context Path

You can check the bindery context path by using the CONFIG command. The Distinguished Names of each effective container are listed on separate lines. When a client searches for a bindery object, it looks through the containers in the order they appear in the list. To find user AnnA above, the client would first search OU=HR for the object. If the client does not find it there, the client then searches OU=Mktg and finds the object

5.3 Bindery Context Eclipsing

The Bindery Context Path can create one potential problem. In NDS, two entries can have the same Relative Distinguished Name. For example, in the figure below, user entries with the RDN of "AnnA" exist both in the OU=HR and OU=Mktg containers.

Figure 5-3 *Bindery Services and Eclipsing*



NDS distinguishes between the two entries because their Distinguished Names are different. However, the bindery client or server sees only the object's Relative Distinguished Name. If a bindery client searches for the object AnnA, it finds the object in OU=HR first and stops searching, whether or not that is the object the client is looking for. In this situation, the client is unable to access the object AnnA in the OU=Mktg container. This effect is called eclipsing.

Eclipsing also occurs when a dynamic bindery object has the same name as a static bindery object, although this situation is rare. In such cases, the dynamic object always eclipses the static object.

You can solve this problem by making sure that no two bindery objects have the same Relative Distinguished Name.

5.4 NDS Bindery Objects

The bindery used in previous versions of NetWare allowed two classes of objects:

- Dynamic. These objects were automatically deleted when the bindery was closed and reopened. They were used primarily for SAP information, so advertising servers would have their objects added to the bindery as dynamic objects. If the bindery were closed and reopened, these objects and their properties were deleted.
- Static. These objects were permanent in the bindery until an administrator deleted them manually.

The bindery services code in NDS provides an alternate view of the same data stored in the NDS Directory Information Base (DIB). Bindery services maps the entries and attributes stored in NDS to the familiar bindery format, for both objects and properties.

Not all information available in the NDS database is visible to bindery APIs.

In NDS Bindery Services, dynamic bindery objects are not stored in normal NDS containers. They do not have a specific bindery parent container but are stored in a dedicated "bindery partition."

They can be accessed by a bindery client or server but are not available from NDS APIs or when the bindery is closed.

NDS has a global Bindery Open Time variable, and only dynamic bindery objects whose creation time is more recent than the Bindery Open Time are valid; the others are invalid and are scheduled to be deleted by the purge process. This makes creating dynamic bindery objects more efficient, since old values that have not been purged can be used when an old object is to be added again, as often happens with SAP data.

Likewise, static bindery objects are stored in normal NDS containers specified in the bindery context path. They are normal NDS objects and are stored persistently in the NDS database, whether or not the bindery is open. When the bindery is open, they are visible to bindery calls.

Undifferentiated properties are attributes of the Bindery Property attribute type. They do not map to NDS attributes because the caller that writes the property value defines the data format. Because property names can conflict, NDS defines the following precedence when searching for bindery properties: Canonized properties first, undifferentiated properties, and finally attribute properties.

The following well-known bindery properties do not map to NDS attributes and are stored as Bindery Property attribute values:

- BLOCKS_READ
- BLOCKS_WRITTEN
- CONNECT_TIME
- DISK_STORAGE
- DOMAIN_NAME
- REQUESTS_MADE

Unlike the Directory tree, which is hierarchical, the bindery is a flat structure that is specific to one server. To provide access for bindery users and clients, NDS imitates a flat structure for leaf entries within one or more container entries.

NDS™ emulates the Bindery found in earlier versions of NetWare® by formulating NDS information into a bindery format that can be used to respond to bindery-style requests. This is necessary for backward compatibility with applications that perform bindery functions.

See Also:

- “Bindery Types” on page 96
- “Bindery Properties” on page 97
- “Item Properties” on page 98
- “Canonized Properties” on page 98
- “Attribute Properties” on page 101
- “Undifferentiated Properties” on page 102

5.4.1 Bindery Types

The hierarchical nature of NDS requires that it distinguish between types of entries, such as container and leaf entries. Consequently, each object must belong to a valid object class specified in the Directory schema. The schema defines what attributes each object class has, thus regulating the function of each object class.

The bindery, however, is not hierarchical and does not need to distinguish objects by class, so it has no formal schema definitions. However, the bindery allowed object types, designated by 16-bit values that grouped similar objects. For example, user objects were type 1, and group objects were type 2.

Some of the well-known object types have direct counterparts in the NDS schema and can be converted, or mutated, from the Bindery Object class to the appropriate NDS object class. For a bindery object type to be mutated, it must have all the mandatory attributes of the target object class and cannot have the same name as another object in the container. Currently, the following bindery object types can be mutated: These object classes, along with the Bindery Object class make up all the objects visible through bindery calls. No other NDS object classes are visible through bindery calls.

Table 5-1 *Bindery Object Type With Corresponding NDS Object Class*

Bindery Object Type	Corresponding NDS Object Class
1	User
2	Group
3	Queue
7	Print Server
309	Profile

5.4.2 Bindery Properties

The NDS schema allows any object class to have a Bindery Property attribute type through a special case check; the schema defines the attribute, but only the Bindery Object class lists it as an optional attribute. This allows for backwards compatibility with bindery-based application that define their own properties.

Although any object class can have a Bindery Property attribute, only the following classes will have the attribute:

- The five object classes listed in "Bindery Types" on page 96
- The Bindery Object class, which is assigned to all other objects created through bindery NCP calls.

Each bindery property can contain two types of data:

- Item property. This property, which consists of 128-byte segments, allows a user to define an object's attributes. This can contain any arbitrary, user-defined data.
- Set. This attribute contains a list of object IDs.

5.4.3 Item Properties

When NDS creates an object using bindery services calls, that object is given the Bindery Object object class. The object also has a Bindery Type attribute that defines what type of bindery object it is. The Bindery Type attribute is a 16-bit integer corresponding to the type field in all calls related to bindery objects. This type has a well-known meaning. Bindery object item properties can be grouped into three categories according to the way NDS handles them:

- Canonized properties. These bindery properties are those commonly used in bindery-based networks, with well-known data formats. They do not map cleanly to Directory Services attributes. These properties could contain several attributes, could have a different data format than the corresponding NDS attribute, or could require a fixed security value. NDS converts these to valid NDS attributes for storage, but APIs still access them as bindery objects.
- Attribute properties. These bindery properties map directly to Directory Services attributes, with similar if not identical data formats.
- Undifferentiated properties. These bindery properties, also known as bag properties, have no direct equivalents in Directory Services. They are of the attribute type Bindery Property.

When a client adds a property to a bindery object, an NDS server calls the routines to add and write an entry and does the following:

- Checks its table of canonized properties to see if it recognizes the attribute being added (see “Canonized Properties” below): If the server finds the property in its canonized properties table, it converts the properties into as many corresponding NDS attributes as necessary, and the operation ends; otherwise, it continues to step 2.
- Checks its table of attribute properties (see “Attribute Properties” below): If the server finds the property in its attribute properties table, it converts the object’s property into a corresponding NDS attribute, and the operation ends; otherwise, it continues to step 3.
- The server labels the property “undifferentiated” and stores the property’s value without attempting to validate the data (see “Undifferentiated Properties” below).

5.4.4 Canonized Properties

Canonized properties are those commonly used in bindery-based networks, with well-known data formats. They do not map cleanly to Directory Services attributes. These properties could contain several attributes, could have a different data format than the corresponding NDS attribute, or could require a fixed security value. NDS converts these to valid NDS attributes for storage, but APIs still access them as bindery objects.

The following bindery properties are canonized for the following NDS object classes.

Bindery Property	NDS Class	NDS Attribute It Maps To
GROUP_MEMBERS (set)	Group	Members (Distinguished Names)
GROUPS_I’M_IN	User	Group Membership
HOME_DIRECTORY	User	Home Directory
MANAGERS (object IDs)	User Group	A list of ACL attributes (one for each ID) that grant NDS Add [Entry Rights] for that ID

Bindery Property	NDS Class	NDS Attribute It Maps To
MISC_LOGIN_INFO	User	Last Login Time
NET_ADDRESS	All	Network Address
NODE_CONTROL	User	Network Address Restriction
OBJ_SUPERVISORS	User Group Profile	A list of ACL attributes (one for each ID) that grant NDS Supervisor [Entry Rights] to that ID
OPERATORS	<Local server>	Operator
PROFILES_I'M_IN	User Group	Profile Membership
PS_OPERATORS	Print Server	Operator
PS_USERS	Print Server	User
Q_DIRECTORY	Queue	Queue Directory
Q_OPERATORS	Queue	Operator
Q_SERVERS	Queue	Server
SECURITY_EQUALS	User	Security Equals
USERS	Profile	A list of ACL attributes (one for each ID) that grant NDS Read [All Attribute Rights] for that ID.

The following canonized properties map to multiple NDS attributes.

LOGIN_CONTROL. The LOGIN_CONTROL bindery property maps to NDS attributes as follows:

Field	Type	NDS Attribute It Maps To
Exp_Date	uint8[3]	Login Expiration Time (year, month, day bytes map to NDS Time syntax).
Acct_Expired	uint8	Login Disabled (boolean).
Pass_Date	uint8[3]	Password Expiration Time (year, month, day bytes map to NDS Time syntax).
Pass_Grace	uint8	Login Grace Remaining (integer).
Exp_Interval	uint16	Password Expiration Interval (days map to NDS Interval syntax).
Grace_Reset	uint8	Login Grace Limit (integer).
Min_Pass_Length	uint8	Password Minimum Length (integer).
Max_Connections	uint16	Login Maximum Simultaneous (integer).

Field	Type	NDS Attribute It Maps To
Time_BitMap	uint8	Login Time:map. **See Allowed Time Map**. The bindery field contains a 42-byte time grid with each bit representing 30 minutes beginning at midnight Sunday. The NDS attribute also contains 42 bytes.
Last_Log_Time	uint8	Login Time
RestrictionFlags	uint8	Bit 1 is PasswordChangeBit, which maps to Password Allow Change. Bit 2 is UniquePasswordBit, which maps to Password Unique Required.
UnknownInfo	uint8	Not mapped.
MaxDiskBlocks	uint32	Not mapped. If read through bindery services, this field always has the value 7fffffffh.
BadLogCount	uint16	Login Intruder Attempts.
NextResetTime	uint8	Login Intruder Reset Time. The bindery value is expressed as an integer value of minutes since 1 January 1985. The NDS value is a Time syntax (seconds since 1 Jan 1970).
BadStnAddress	uint8[12]	Login Intruder Address. The bindery syntax is a network address, Net:Node:Socket. The NDS syntax is a Network Address

The ACCOUNT_BALANCE bindery property maps to NDS attributes as follows:

Field	Type	NDS Attribute It Maps To
balance	long	Account Balance
creditLimit	long	If the value is 8000000h, this field maps to Allow Unlimited Credit (True). Otherwise, its value is False and the field contains a value for the Minimum Account Balance.
reserved	char	Not mapped.

The ACCOUNT_HOLD bindery property maps to NDS properties as follows:

Field	Type	NDS Attribute It Maps To
ServerHoldStruct	long serverID long holdAmount	Server Holds. Each valid entry in the bindery property becomes a separate value. The bindery property holds an array of up to 16 ID-amount pairs, each of which becomes a separate value for the User’s multivalued Server Holds attribute.

The ACCT_LOCKOUT field implies that the NDS attribute Detect Intruders is set to True. The ACCT_LOCKOUT bindery property maps to NDS attributes as follows:

Field	Type	NDS Attribute It Maps To
intruderCount	short	Login Intruder Limit
intruderResetInterval	short	Intruder Attempt Reset Interval. However, the bindery value is in seconds, while the NDS value is in minutes.
intruderLockoutInterval	short	Intruder Lockout Reset Interval

The DEFAULT_PROFILE bindery property maps to NDS attributes as follows:

Field	Type	NDS Attribute It Maps To
Profile Name	char	Profile (DN)

Additionally, the Password bindery property is handled in a special case and maintained in the NDS Private Key attribute.

5.4.5 Attribute Properties

These bindery properties map directly to Directory Services attributes, with similar if not identical data formats. Attribute properties names longer than the 15-character limit for bindery properties may have alternate names when viewed through bindery services calls; in some cases, the alternate name is completely different than the original property name.

Table 5-2 *Mapping Details of Attribute Properties*

Bindery Property	NDS Attribute It Maps To
DELIVERY_OFFICE	Physical Delivery Office Name
EQUAL _TO_ME	Equivalent To Me
FAX_NUMBER	Facsimile Telephone Number
HOST_RESOURCE)	Host Resource Name
IDENTIFICATION	Full Name
MISC_LOGIN_INFO	Last Login Time
PAGE_LANGUAGE	Page Description Language
PHONE_NUMBER	Telephone Number
POST_OFFICE_BOX	Postal Office Box
RESTRICTION	Bindery Object Restriction
SERVICES	Supported Services

5.4.6 Undifferentiated Properties

These bindery properties map directly to Directory Services attributes, with similar if not identical data formats. Attribute properties names longer than the 15-character limit for bindery properties may have alternate names when viewed through bindery services calls; in some cases, the alternate name is completely different than the original property name.

Table 5-3 *Mapping Details of Undifferentiated Propertiess*

Bindery Property	**NDS Attribute It Maps To**
DELIVERY_OFFICE	Physical Delivery Office Name
EQUAL_TO_ME	Equivalent To Me
FAX_NUMBER	Facsimile Telephone Number
HOST_RESOURCE)	Host Resource Name
IDENTIFICATION	Full Name
MISC_LOGIN_INFO	Last Login Time
PAGE_LANGUAGE	Page Description Language
PHONE_NUMBER	Telephone Number
POST_OFFICE_BOX	Postal Office Box
RESTRICTION	Bindery Object Restriction
SERVICES	Supported Services

NDS Return Values

6

This chapter lists the values that NDS can return. The values from 1 to 255 are values that can be returned from the underlying operating system, usually NetWare. The values in the 300 range are values that can be returned by the NDS client. The values over 600 are values that can be returned by the NDS agent.

6.1 NDS Return Values from the Operating System

The NDS return values are defined in the nwdserr.h file. The table below lists the errors numerically by their negative decimal values.

Table 6-1 *NDS Return Values from the Operating system*

Hex	Dec	Constant: Description
0xFFFF FFFF	-001	DSERR_INSUFFICIENT_SPACE
0xFFFF FF89	-119	DSERR_BUFFER_TOO_SMALL: The data to be passed back is too large for the buffer you have declared
0xFFFF FF88	-120	DSERR_VOLUME_FLAG_NOT_SET
0xFFFF FF87	-121	DSERR_NO_ITEMS_FOUND: You made a bindery request for items not found
0xFFFF FF86	-122	DSERR_CONN_ALREADY_TEMPORARY: Attempted to convert a temporary connection into a temporary connection
0xFFFF FF85	-123	DSERR_CONN_ALREADY_LOGGED_IN: Attempted to log in to a server you were already logged into
0xFFFF FF84	-124	DSERR_CONN_NOT_AUTHENTICATED: Attempted connection for call without being authenticated
0xFFFF FF83	-125	DSERR_CONN_NOT_LOGGED_IN: Attempted to log out of a connection you are not logged into
0xFFFF FF82	-126	DSERR_NCP_BOUNDARY_CHECK_FAILED: NCP subfunction size does not match the actual size of data sent
0xFFFF FF81	-127	DSERR_LOCK_WAITING: Time-out occurred before file was locked
0xFFFF FF80	-128	DSERR_LOCK_FAIL: Attempted to open or create a file that is already open
0xFFFF FF7F	-129	DSERR_OUT_OF_HANDLES: No more file handles available; the network file handle table is full
0xFFFF FF7E	-130	DSERR_NO_OPEN_PRIVILEGE: Attempted to open a file without the open privilege
0xFFFF FF7D	-131	DSERR_HARD_IO_ERROR: Hard disk input/output error on a NetWare® volume; a bad sector has been encountered and could be fatal

Hex	Dec	Constant: Description
0xFFFF FF7C	-132	DSERR_NO_CREATE_PRIVILEGE: Attempted to create a file without the create privilege
0xFFFF FF7B	-133	DSERR_NO_CREATE_DELETE_PRIV: Attempted to create an already existing file without the create/delete privileges
0xFFFF FF7A	-134	DSERR_R_O_CREATE_FILE: Attempted to create a file with the same name as an already existing file with read-only status
0xFFFF FF79	-135	DSERR_CREATE_FILE_INVALID_NAME: A file name contains invalid characters
0xFFFF FF78	-136	DSERR_INVALID_FILE_HANDLE: Attempted to close or perform I/O on a file with an invalid file handle (i.e. trying to read from a file that has been closed)
0xFFFF FF77	-137	DSERR_NO_SEARCH_PRIVILEGE: Attempted to search a directory without search privileges in that directory
0xFFFF FF76	-138	DSERR_NO_DELETE_PRIVILEGE: Attempted to delete a file without file deletion privileges in that file’s directory
0xFFFF FF75	-139	DSERR_NO_RENAME_PRIVILEGE: Attempted to rename a file without renaming privileges in that file’s directory
0xFFFF FF74	-140	DSERR_NO_SET_PRIVILEGE Attempted to modify a file without attribute modification privileges in that file’s directory
0xFFFF FF73	-141	DSERR_SOME_FILES_IN_USE: Attempted to delete, rename, or set file attributes using an ambiguous filename while some of the files are in use by another workstation
0xFFFF FF72	-142	DSERR_ALL_FILES_IN_USE: Attempted to delete, rename, or set file attributes using a filename when the file or files are in use by another workstation
0xFFFF FF71	-143	DSERR_SOME_READ_ONLY: Attempted to open read-only files
0xFFFF FF70	-144	DSERR_ALL_READ_ONLY: Attempted to delete, rename, or set file attributes using a filename when all of the files specified have read-only status
0xFFFF FF6F	-145	DSERR_SOME_NAMES_EXIST: Attempted to rename files using an ambiguous filename, when one or more files matching the new filename specification already exist
0xFFFF FF6E	-146	DSERR_ALL_NAMES_EXIST: Attempted to rename a file using a filename, when all of the files matching the new filename specification already exist
0xFFFF FF6D	-147	DSERR_NO_READ_PRIVILEGE: Attempted to read a file without read privileges for that file
0xFFFF FF6C	-148	DSERR_NO_WRITE_PRIVILEGE: Attempted to write to a file without file write privileges, or the specified file has read-only status
0xFFFF FF6B	-149	DSERR_FILE_DETACHED: Attempted to read or write to a detached file
0xFFFF FF6A	-150	DSERR_NO_ALLOC_SPACE: Attempted to write to a server that does not currently have enough free DRAM to process this request
0xFFFF FF6A	-150	DSERR_TARGET_NOT_A_SUBDIR
0xFFFF FF6A	-150	ERR_INSUFFICIENT_MEMORY

Hex	Dec	Constant: Description
0xFFFF FF69	-151	DSERR_NO_SPOOL_SPACE: The network OS has determined that the network disk doesn't have enough space left for spool files
0xFFFF FF68	-152	DSERR_INVALID_VOLUME: The network OS cannot find the requested volume in the system definition files
0xFFFF FF67	-153	DSERR_DIRECTORY_FULL: Attempted to write to a volume without available directory space
0xFFFF FF66	-154	DSERR_RENAME_ACROSS_VOLUME: Attempted to rename a file and move it from its current volume into another volume. The rename command may move the file between directories on the same volume; however, using rename to move a file between volumes is not allowed.
0xFFFF FF65	-155	DSERR_BAD_DIR_HANDLE: Attempted to use an invalid directory handle (occurs if the network has been brought down and back up without rebooting the workstation)
0xFFFF FF64	-156	DSERR_INVALID_PATH
0xFFFF FF64	-156	DSERR_NO_SUCH_EXTENSION: No more trustees are listed in the directory
0xFFFF FF63	-157	DSERR_NO_DIR_HANDLES: No more directory handles available. The directory handle table is full. Each user may have up to 255 directory handles.
0xFFFF FF62	-158	DSERR_BAD_FILE_NAME: Attempted to create a file using invalid file name characters
0xFFFF FF61	-159	DSERR_DIRECTORY_ACTIVE: Attempted to delete a directory that is being used by another workstation
0xFFFF FF60	-160	DSERR_DIRECTORY_NOT_EMPTY: Attempted to delete a directory that contains files or other directories
0xFFFF FF5F	-161	DSERR_DIRECTORY_IO_ERROR: A non-recoverable I/O error has occurred on the disk in the directory area (occurred in both copies of the directory and is fatal)
0xFFFF FF5E	-162	DSERR_IO_LOCKED: Attempted to read a file where data is physically locked
0xFFFF FF5D	-163	DSERR_TRANSACTION_RESTARTED
0xFFFF FF5C	-164	DSERR_RENAME_DIR_INVALID
0xFFFF FF5B	-165	DSERR_INVALID_OPENCREATE_MODE
0xFFFF FF5A	-166	DSERR_ALREADY_IN_USE
0xFFFF FF59	-167	DSERR_INVALID_RESOURCE_TAG
0xFFFF FF58	-168	DSERR_ACCESS_DENIED
0xFFFF FE44	-188	DSERR_LOGIN_SIGNING_REQUIRED
0xFFFF FE43	-189	DSERR_LOGIN_ENCRYPT_REQUIRED
0xFFFF FF42	-190	DSERR_INVALID_DATA_STREAM
0xFFFF FF41	-191	DSERR_INVALID_NAME_SPACE

Hex	Dec	Constant: Description
0xFFFF FF40	-192	DSERR_NO_ACCOUNTING_PRIVILEGES
0xFFFF FF3F	-193	DSERR_NO_ACCOUNT_BALANCE: Attempted to log in by a bindery object without an accounting balance while accounting was enabled
0xFFFF FF3E	-194	DSERR_CREDIT_LIMIT_EXCEEDED: Attempted to log in to account with no credit available
0xFFFF FF3D	-195	DSERR_TOO_MANY_HOLDS
0xFFFF FF3C	-196	DSERR_ACCOUNTING_DISABLED
0xFFFF FF3B	-197	DSERR_LOGIN_LOCKOUT: Attempted to log in after the system had locked the account because of intruder detection
0xFFFF FF3A	-198	DSERR_NO_CONSOLE_RIGHTS: Attempted to use console without operator privileges
0xFFFF FF30	-208	DSERR_Q_IO_FAILURE
0xFFFF FF2F	-209	DSERR_NO_QUEUE
0xFFFF FF2E	-210	DSERR_NO_Q_SERVER
0xFFFF FF2D	-211	DSERR_NO_Q_RIGHTS
0xFFFF FF2C	-212	DSERR_Q_FULL
0xFFFF FF2B	-213	DSERR_NO_Q_JOB
0xFFFF FF2A	-214	DSERR_NO_Q_JOB_RIGHTS
0xFFFF FF2A	-214	DSERR_UNENCRYPTED_NOT_ALLOWED
0xFFFF FF29	-215	DSERR_Q_IN_SERVICE
0xFFFF FF29	-215	DSERR_DUPLICATE_PASSWORD: Attempted to change password to a previously used password when the unique requirement is specified for the account
0xFFFF FF28	-216	DSERR_Q_NOT_ACTIVE
0xFFFF FF28	-216	DSERR_PASSWORD_TOO_SHORT: Attempted to change password to one with fewer characters than the required minimum specified for the account
0xFFFF FF27	-217	DSERR_Q_STN_NOT_SERVER
0xFFFF FF27	-217	DSERR_MAXIMUM_LOGINS_EXCEEDED: Attempted to log in using an account that has limits on the number of concurrent connections and that number has been reached
0xFFFF FF26	-218	DSERR_Q_HALTED
0xFFFF FF26	-218	DSERR_BAD_LOGIN_TIME: Attempted to log in during an unauthorized time of day specified for the account
0xFFFF FF25	-219	DSERR_Q_MAX_SERVERS
0xFFFF FF25	-219	DSERR_NODE_ADDRESS_VIOLATION: Attempted to log in from an unauthorized station using an account with limits to a specific network and/or station

Hex	Dec	Constant: Description
0xFFFF FF24	-220	DSERR_LOG_ACCOUNT_EXPIRED: Attempted to log in using an account that has expired or has been disabled by the Supervisor
0xFFFF FF22	-222	DSERR_BAD_PASSWORD: Attempted to log in using an account password that has expired and all grace logins have also expired
0xFFFF FF21	-223	DSERR_PASSWORD_EXPIRED: Attempted to log in using an expired account password but the login was allowed because the account had a grace login
0xFFFF FF20	-224	DSERR_NO_LOGIN_CONN_AVAILABLE
0xFFFF FF18	-232	DSERR_WRITE_TO_GROUP_PROPERTY: Attempted to write a data segment to a group property using the call to write a property value
0xFFFF FF17	-233	DSERR_MEMBER_ALREADY_EXISTS: Attempted to redundantly add an object to a group property
0xFFFF FF16	-234	DSERR_NO_SUCH_MEMBER: Attempted to use an object that is not a member of the defined group property
0xFFFF FF15	-235	DSERR_PROPERTY_NOT_GROUP: Attempted to use a non-group property
0xFFFF FF14	-236	DSERR_NO_SUCH_VALUE_SET: Attempted to use a nonexistent segment. Segments must be written sequentially when a property is first created, but may be read and written in any order once they exist.
0xFFFF FF13	-237	DSERR_PROPERTY_ALREADY_EXISTS
0xFFFF FF12	-238	DSERR_OBJECT_ALREADY_EXISTS
0xFFFF FF11	-239	DSERR_ILLEGAL_NAME: Request made with an object or property name containing illegal characters (control characters, comma, colon, semicolon, slash, backslash, question mark, asterisk, and tilde)
0xFFFF FF10	-240	DSERR_ILLEGAL_WILDCARD: Attempted to use a wildcard character or wild object type in a call where wildcards are not allowed
0xFFFF FF0F	-241	DSERR_BINDERY_SECURITY: Attempted to assign a security level of a bindery object or property to be higher than the user's security level (makes the object or property inaccessible to the user)
0xFFFF FF0E	-242	DSERR_NO_OBJECT_READ_RIGHTS: Attempted to access object information or scan the object properties using a station without the necessary security to access this information
0xFFFF FF0D	-243	DSERR_NO_OBJECT_RENAME_RIGHTS: Attempted to rename an object without the necessary security. Only the Supervisor can rename objects. If the station does not have the proper security to see that the object exists, NCP_NO_SUCH_OBJECT is returned.
0xFFFF FF0C	-244	DSERR_NO_OBJECT_DELETE_RIGHTS: Attempted to delete an object using a station without the necessary security to delete the object. Only the Supervisor can delete objects. If the station does not even have the proper security to see that the object exists, NCP_NO_SUCH_OBJECT is returned.
0xFFFF FF0B	-245	DSERR_NO_OBJECT_CREATE_RIGHTS: Attempted to create an object using a station without the necessary security to create or change an object (only Supervisors are allowed to create objects)

Hex	Dec	Constant: Description
0xFFFF FF0A	-246	DSERR_NO_PROPERTY_DELETE_RIGHTS: Attempted to delete a property using a station without the necessary security privilege to delete a property from the given object. If the station does not have the proper security to see that the property exists, NCP_NO_SUCH_PROPERTY is returned.
0xFFFF FF09	-247	DSERR_NO_PROPERTY_CREATE_RIGHTS: Attempted to create a property using a station without the necessary security to create or change a property for the object
0xFFFF FF08	-248	DSERR_NO_PROPERTY_WRITE_RIGHTS
0xFFFF FF07	-249	DSERR_NO_PROPERTY_READ_RIGHTS: Attempted to read property data using a station without the necessary rights to access the property data
0xFFFF FF06	-250	DSERR_TEMP_REMAP: Attempted to use an unknown path
0xFFFF FF05	-251	DSERR_UNKNOWN_REQUEST: Attempted to use an invalid parameter (drive number, path, or flag value) during a set drive path call
0xFFFF FF05	-251	DSERR_NO_SUCH_PROPERTY: Attempted to use a property that does not exist for the specified object
0xFFFF FF04	-252	DSERR_MESSAGE_QUEUE_FULL
0xFFFF FF04	-252	DSERR_TARGET_ALREADY_HAS_MSG
0xFFFF FF04	-252	DSERR_NO_SUCH_OBJECT: Attempted to use an object that doesn’t exist, or the calling station doesn’t have the proper security to access the object. The object name and type must both match for the object to be found.
0xFFFF FF03	-253	DSERR_BAD_STATION_NUMBER: Attempted to use a bad (undefined, unavailable, etc.) station number
0xFFFF FF02	-254	DSERR_BINDERY_LOCKED: Attempted to use a bindery that is temporarily locked by the Supervisor
0xFFFF FF02	-254	DSERR_DIR_LOCKED
0xFFFF FF02	-254	DSERR_SPOOL_DELETE
0xFFFF FF02	-254	DSERR_TRUSTEE_NOT_FOUND
0xFFFF FF02	-254	DSERR_TIMEOUT
0xFFFF FF01	-255	DSERR_HARD_FAILURE
0xFFFF FF01	-255	DSERR_FILE_NAME
0xFFFF FF01	-255	DSERR_FILE_EXISTS
0xFFFF FF01	-255	DSERR_CLOSE_FCB
0xFFFF FF01	-255	DSERR_IO_BOUND: Attempted to write beyond the end of the file or disk
0xFFFF FF01	-255	DSERR_NO_SPOOL_FILE
0xFFFF FF01	-255	DSERR_BAD_SPOOL_PRINTER: Attempted to use a bad (undefined, unavailable, etc.) printer
0xFFFF FF01	-255	DSERR_BAD_PARAMETER

Hex	Dec	Constant: Description
0xFFFF FF01	-255	DSERR_NO_FILES_FOUND: No files were found matching the search specification
0xFFFF FF01	-255	DSERR_NO_TRUSTEE_CHANGE_PRIV
0xFFFF FF01	-255	DSERR_TARGET_NOT_LOGGED_IN
0xFFFF FF01	-255	DSERR_TARGET_NOT_ACCEPTING_MSGS
0xFFFF FF01	-255	DSERR_MUST_FORCE_DOWN
0xFFFF FF01	-255	DSERR_OF_SOME_SORT

6.2 NDS Client Return Values

The NDS return values are defined in the nwdserr.h file. The table below lists the errors numerically by their negative decimal values.

Table 6-2 *NDS Client Return Values*

Hex	Dec	Constant: Description
0xFFFF FED3	-301	ERR_NOT_ENOUGH_MEMORY: Client workstation does not have memory to allocate
0xFFFF FED2	-302	ERR_BAD_KEY: Trying to pass a bad key parameter for a context call (see NWDSDC.H for the correct parameter).
0xFFFF FED1	-303	ERR_BAD_CONTEXT: Trying to pass a bad context parameter to a NDS function. Call NWDSCreateContext first and use its return value as the context parameter.
0xFFFF FED0	-304	ERR_BUFFER_FULL: Ran out of room trying to add data to an input buffer
0xFFFF FECF	-305	ERR_LIST_EMPTY: Passed an empty list (a null pointer) to NWDSPutAttrVal for the SYN_CI_LIST or SYN_OCTET_LIST syntax type
0xFFFF FECE	-306	ERR_BAD_SYNTAX: Tried to pass a bad syntax ID
0xFFFF FECD	-307	ERR_BUFFER_EMPTY: Tried to get data from an empty buffer
0xFFFF FECC	-308	ERR_BAD_VERB: Initialized the buffer with a verb not associated with the API call
0xFFFF FECB	-309	ERR_EXPECTED_IDENTIFIER: The name being parsed is not typed
0xFFFF FECA	-310	ERR_EXPECTED_EQUALS: An equal sign is expected in the name
0xFFFF FEC9	-311	ERR_ATTR_TYPE_EXPECTED: The name being parsed is a multi-AVA and must be typed (all AVAs must be either typed or not typed)
0xFFFF FEC8	-312	ERR_ATTR_TYPE_NOT_EXPECTED: The name being parsed is a multi-AVA and must not be typed (all AVAs must be either typed or not typed)
0xFFFF FEC7	-313	ERR_FILTER_TREE_EMPTY: Tried to delete an empty filter

Hex	Dec	Constant: Description
0xFFFF FEC6	-314	ERR_INVALID_OBJECT_NAME: (1) Tried to pass a NULL string for object name to the API call or (2) Tried to pass a name containing both leading and trailing dots
0xFFFF FEC5	-315	ERR_EXPECTED_RDN_DELIMITER: An RDN delimiter (.) was expected and not found during the name parse
0xFFFF FEC4	-316	ERR_TOO_MANY_TOKENS: Too many trailing delimiter dots in name; only three context levels and four trailing dots in name are permitted
0xFFFF FEC3	-317	ERR_INCONSISTENT_MULTIAVA: AVA type passed in is wrong
0xFFFF FEC2	-318	ERR_COUNTRY_NAME_TOO_LONG: Country name identifiers are only allowed one character
0xFFFF FEC1	-319	ERR_SYSTEM_ERROR: Internal error
0xFFFF FEC0	-320	ERR_CANT_ADD_ROOT: Tried to restore an object at the root
0xFFFF FEBF	-321	ERR_UNABLE_TO_ATTACH: Could not connect to the specified server
0xFFFF FEBE	-322	ERR_INVALID_HANDLE: Invalid iteration handle
0xFFFF FEBD	-323	ERR_BUFFER_ZERO_LENGTH: Tried to call NWDSAllocBuf with a zero-length size
0xFFFF FEBC	-324	ERR_INVALID_REPLICA_TYPE: Attempted to pass in a replica type that was not a MASTER, SECONDARY, or READONLY
0xFFFF FEBB	-325	ERR_INVALID_ATTR_SYNTAX: Attempted to pass in an invalid attribute syntax ID
0xFFFF FEBA	-326	ERR_INVALID_FILTER_SYNTAX: Attempted to pass in an invalid filter syntax ID
0xFFFF FEB8	-328	ERR_CONTEXT_CREATION: Failed to create a context-usually because Unicode tables were not loaded
0xFFFF FEB7	-329	ERR_INVALID_UNION_TAG: The server returned an infoType parameter that did not agree with the infoType you passed in
0xFFFF FEB6	-330	ERR_INVALID_SERVER_RESPONSE: Returned from NWDSGetSyntaxID
0xFFFF FEB5	-331	ERR_NULL_POINTER: Entered a NULL pointer, a real pointer was expected
0xFFFF FEB4	-332	ERR_NO_SERVER_FOUND
0xFFFF FEB3	-333	ERR_NO_CONNECTION: Internal error-contact Novell® Customer Support
0xFFFF FEB2	-334	ERR_RDN_TOO_LONG: The RDN exceeded 128 characters
0xFFFF FEB1	-335	ERR_DUPLICATE_TYPE: Multi-AVAs-AVAs cannot contain same type
0xFFFF FEB0	-336	ERR_DATA_STORE_FAILURE
0xFFFF FEAF	-337	ERR_NOT_LOGGED_IN
0xFFFF FEAE	-338	ERR_INVALID_PASSWORD_CHARS: Entered password characters that are invalid
0xFFFF FEAD	-339	ERR_FAILED_SERVER_AUTHENT: Attempted server authentication failed
0xFFFF FEAC	-340	ERR_TRANSPORT: Transport failed

Hex	Dec	Constant: Description
0xFFFF FEAB	-341	ERR_NO_SUCH_SYNTAX: Attempted to use an invalid syntax
0xFFFF FEAA	-342	ERR_INVALID_DS_NAME: (1) An empty string passed in for a name or (2) a NULL pointer
0xFFFF FEA9	-343	ERR_ATTR_NAME_TOO_LONG: Attribute name exceeded 32 characters
0xFFFF FEA8	-344	ERR_INVALID_TDS: Tagged Data Store is either uninitialized or corrupted. Usually, NWDSLogin was not called first.
0xFFFF FEA7	-345	ERR_INVALID_DS_VERSION
0xFFFF FEA6	-346	ERR_UNICODE_TRANSLATION: A unicode translation error returned from one of three functions: NWDSListPartitions, NWDSSyncPartition, and NWDSSyncSchema
0xFFFF FEA5	-347	ERR_SCHEMA_NAME_TOO_LONG: Schema name exceeded 32 characters
0xFFFF FEA4	-348	ERR_UNICODE_FILE_NOT_FOUND: Unicode file could not be found in the defined search algorithm defined in NWInitUnicodeTables
0xFFFF FEA3	-349	ERR_UNICODE_ALREADY_LOADED: (DOS-only) NWInitUnicodeTables attempted to call unicode tables more than once
0xFFFF FEA2	-350	ERR_NOT_CONTEXT_OWNER
0xFFFF FEA1	-351	ERR_ATTEMPT_TO_ATHENTICATE_0
0xFFFF FEA0	-352	ERR_NO_WRITABLE_REPLICAS: Returned by NWDSLogout: On logout, the server logs out of the monitor connection. Subsequently the API call tries to find a writable replica of that monitor connection's partition but cannot.
0xFFFF FE9F	-353	ERR_DN_TOO_LONG: The name passed in exceeded 256 characters
0xFFFF FE9E	-354	ERR_RENAME_NOT_ALLOWED: Attempt to move an object to the same place in the tree that it was in (see NWDSMoveObject)
0xFFFF FE9D	-355	ERR_NOT_NDS_FOR_NT
0xFFFF FE9C	-356	ERR_NDS_FOR_NT_NO_DOMAIN
0xFFFF FE9B	-357	ERR_NDS_FOR_NT_SYNC_DISABLED
0xFFFF FE9A	-358	ERR_ITR_INVALID_HANDLE: Attempted to pass a parameter that was not a valid iterator object.
0xFFFF FE99	-359	ERR_ITR_INVALID_POSITION: Attempted to position the iterator in a logical position that is not within the 0 to 1000 range.
0xFFFF FE98	-360	ERR_ITR_INVALID_SEARCH_DATA: Indicates that the entry data is in an unexpected format.
0xFFFF FE97	-361	ERR_ITR_INVALID_SCOPE: Attempted to specify a subtree search which is not supported on iterator objects.
0xFFFF FE96	-362	ERR_ITR_MAX_COUNT: Indicates that the caller set a limit on the number of entries to count before returning and that this limit has been reached.

6.3 NDS Agent Return Values

The NDS return values are defined in the nwdserr.h file. The table below lists the errors numerically by their negative decimal values.

Table 6-3 *NDS Agent Return Values*

Hex	Dec	Constant: Description
0xFFFF FDA7	-601	ERR_NO_SUCH_ENTRY: Object passed in could not be found.Either the object does not exist on the replying server or the requester has insufficient rights to the object. Action: Check the context relative to the passed-in name.
0xFFFF FDA6	-602	ERR_NO_SUCH_VALUE: The requested attribute value could not be found. Either the object does not have this value for the attribute or the requester has insufficient rights to the object's attributes.
0xFFFF FDA5	-603	ERR_NO_SUCH_ATTRIBUTE: The requested attribute could not be found. Either the specified attribute definition does not exist on the replying server or the specified object class does not include this attribute.
0xFFFF FDA4	-604	ERR_NO_SUCH_CLASS: The class does not exist.
0xFFFF FDA3	-605	ERR_NO_SUCH_PARTITION: The specified partition does not exist on the server replying to the request. This error may indicate inconsistent data in the local database that may be repaired with DSRepair.
0xFFFF FDA2	-606	ERR_ENTRY_ALREADY_EXISTS: Attempted to create, rename, or restore an object when an object with the same RDN already exists in that container in the NDS tree.
0xFFFF FDA1	-607	ERR_NOT_EFFECTIVE_CLASS: Attempted to create an object of a base class that is not an effective class
0xFFFF FDA0	-608	ERR_ILLEGAL_ATTRIBUTE: Attempted to add an attribute to an object when the attribute is not in the list of optional or mandatory attributes for the object's expanded class definition.
0xFFFF FD9F	-609	ERR_MISSING_MANDATORY: Attempted to add an object without specifying values for all of the object's mandatory attributes.
0xFFFF FD9E	-610	ERR_ILLEGAL_DS_NAME: Either the DN or RDN was improperly formatted or the DN exceeded the maximum number of characters (256 is the current maximum).
0xFFFF FD9D	-611	ERR_ILLEGAL_CONTAINMENT: Attempted to add an object or class definition that violates the schema's containment rules.
0xFFFF FD9C	-612	ERR_CANT_HAVE_MULTIPLE_ VALUES: Attempted to add more than one value to a single-value attribute
0xFFFF FD9B	-613	ERR_SYNTAX_VIOLATION: Attempted to modify an attribute using data that does not conform to the syntax specified for the attribute in the schema.
0xFFFF FD9A	-614	ERR_DUPLICATE_VALUE: Attempted to add the same attribute-value combination to an object without overwriting or deleting the existing value.

Hex	Dec	Constant: Description
0xFFFF FD99	-615	ERR_ATTRIBUTE_ALREADY_EXISTS: Attempted to add an attribute definition when an attribute with the same name already exists.
0xFFFF FD98	-616	ERR_MAXIMUM_ENTRIES_EXIST: The server has reached the maximum entries in its database (in NetWare 4.x and 5.0, this is 16, 777, 215).
0xFFFF FD97	-617	ERR_DATABASE_FORMAT: The record structure of the NDS database does not match the structure expected by the version of the DS.NLM. For resolution of this error, contact your Novell Authorized Service Center.
0xFFFF FD96	-618	ERR_INCONSISTENT_DATABASE: The server has detected an inconsistency in the local database. Action: Run DSRepair.
0xFFFF FD95	-619	ERR_INVALID_COMPARISON: Attempted to (1) compare two attributes that are not comparable or (2) use an invalid compare syntax
0xFFFF FD94	-620	ERR_COMPARISON_FAILED: The two attributes submitted for comparison do not match based on the submitted comparison criteria.
0xFFFF FD93	-621	ERR_TRANSACTIONS_DISABLED: TTS was disabled for the server on which the NDS operation is taking place and therefore NDS is unable to service the request. Action: Enable TTS.
0xFFFF FD92	-622	ERR_INVALID_TRANSPORT: The type of transport passed in to the server is not supported by the server
0xFFFF FD91	-623	ERR_SYNTAX_INVALID_IN_NAME: Attempted to create a class definition that specifies a naming attribute whose schema definition is not constrained as a string.
0xFFFF FD90	-624	ERR_REPLICA_ALREADY_EXISTS: Attempted to add a replica when the server already holds a replica of that partition.
0xFFFF FD8F	-625	ERR_TRANSPORT_FAILURE: The source server is unable to communicate with the target server.
0xFFFF FD8E	-626	ERR_ALL_REFERRALS_FAILED: Server has no objects matching request and has attempted to contact x other servers to find the object. Not one of those servers has responded.
0xFFFF FD8D	-627	ERR_CANT_REMOVE_NAMING_ VALUE: Attempted to delete the naming attribute. Rename the object, then delete the attribute value.
0xFFFF FD8C	-628	ERR_OBJECT_CLASS_VIOLATION: Attempted to add an object without specifying the object's base class.
0xFFFF FD8B	-629	ERR_ENTRY_IS_NOT_LEAF: Attempted to delete an entry containing subordinates, which the API call cannot do. First delete the subordinates.
0xFFFF FD8A	-630	ERR_DIFFERENT_TREE: Attempted to submit a request to a server that exists in a different NDS tree.
0xFFFF FD89	-631	ERR_ILLEGAL_REPLICA_TYPE: Attempted to perform a replica operation on a replica that is the wrong type. For example, this error is returned when an attempt is made to set the bindery context on a server which does not have a Read/Write replica of the container referenced in the bindery context.

Hex	Dec	Constant: Description
0xFFFF FD88	-632	ERR_SYSTEM_FAILURE: Unexpected results have occurred. For example, the client requested a network address attribute and NDS returned a public key attribute. The condition can be temporary. If not, run DSRepair.
0xFFFF FD87	-633	ERR_INVALID_ENTRY_FOR_ROOT: Attempted one of the following: (1) to restore or move an entry that is flagged as a partition root but its base class is not a container or (2) to split a partition at an entry whose base class is not a container.
0xFFFF FD86	-634	ERR_NO_REFERRALS: Server has no objects that match request and has no referrals to servers holding the object's replica.
0xFFFF FD85	-635	ERR_REMOTE_FAILURE: Attempt to connect to remote server failed.
0xFFFF FD84	-636	ERR_UNREACHABLE_SERVER: Attempted to perform a partition operation, but one of the servers in the replica ring is unavailable due to ERR_TRANSPORT_FAILURE. Action: Resolve the communication error between the two servers.
0xFFFF FD83	-637	ERR_PREVIOUS_MOVE_IN_PROGRESS: Attempted to move an entry a second time, before the first context move has been synchronized to all replicas. Synchronization time depends upon the size of the replica, the number of replicas, and the communication links between the servers holding the replicas.
0xFFFF FD82	-638	ERR_NO_CHARACTER_MAPPING: Attempted to map a character in a specified RDN or DN to the Unicode representation, and the mapping failed.
0xFFFF FD81	-639	ERR_INCOMPLETE_AUTHENTICATION: An unexpected error occurred during the final steps of authentication. This is usually caused by packets corrupted by faulty LAN drivers, LAN cards, or routers.
0xFFFF FD80	-640	ERR_INVALID_CERTIFICATE: Not currently used.
0xFFFF FD7F	-641	ERR_INVALID_REQUEST: Server did not understand request-for example, the client could have sent an incorrect verb for the NDS version on the server.
0xFFFF FD7E	-642	ERR_INVALID_ITERATION: Iteration handle passed in by client is wrong. This is usually caused by packets corrupted by faulty LAN drivers, LAN cards, or routers.
0xFFFF FD7D	-643	ERR_SCHEMA_IS_NONREMOVABLE: Attempted to delete an operational class definition or attribute definition which has been flagged as nonremovable.
0xFFFF FD7C	-644	ERR_SCHEMA_IS_IN_USE: Attempted to delete a class definition that is used in the expanded class definition of an object, that is specified in the containment list of another schema class definition, or that is currently used by an object in the database.
0xFFFF FD7B	-645	ERR_CLASS_ALREADY_EXISTS: Attempted to add a class definition when a class definition with the same name already exists.
0xFFFF FD7A	-646	ERR_BAD_NAMING_ATTRIBUTES: Attempted to create a class definition whose naming attributes list contains an attribute that is not specified as an optional or mandator attribute for the class or its super classes.
0xFFFF FD79	-647	ERR_NOT_ROOT_PARTITION: Attempted to perform a partition operation with a specified object which is not a partition root.

Hex	Dec	Constant: Description
0xFFFF FD78	-648	ERR_INSUFFICIENT_STACK: Not currently used.
0xFFFF FD77	-649	ERR_INSUFFICIENT_BUFFER: A request was received where the buffer provided by the request is too small for the requested data.
0xFFFF FD76	-650	ERR_AMBIGUOUS_CONTAINMENT: Attempted to create an effective class definition when the class does not specify containment and the super classes provide either no containment rules or conflicting containment rules.
0xFFFF FD75	-651	ERR_AMBIGUOUS_NAMING: Attempted to create a class definition when the class does not specify naming attributes and the super classes don't provide any either.
0xFFFF FD74	-652	ERR_DUPLICATE_MANDATORY: Attempted to create a class definition when both the class and one of its super classes list the same attribute as a mandatory naming attribute.
0xFFFF FD73	-653	ERR_DUPLICATE_OPTIONAL: Attempted to create a class definition when both the class and one of its super classes list the same attribute as an optional naming attribute.
0xFFFF FD72	-654	ERR_PARTITION_BUSY: Attempted to start another partition operation while a partition operation is currently taking place. Action: Wait for the previous partition operation to synchronize with all replicas in the ring.
0xFFFF FD71	-655	ERR_MULTIPLE_REPLICAS: An internal error caused by a programming error, not a system inconsistency.
0xFFFF FD70	-656	ERR_CRUCIAL_REPLICA: Attempted an illegal operation on a master replica such as removing it, changing its type, adding another replica on the server where the master of that replica already exists, or requesting that the master receive all updates.
0xFFFF FD6F	-657	ERR_SCHEMA_SYNC_IN_PROGRESS: Function could not be completed because schema synchronization was in progress. In NDS.NLM v5.95 or higher, the schema is locked during synchronization.
0xFFFF FD6E	-658	ERR_SKULK_IN_PROGRESS: Not currently used.
0xFFFF FD6D	-659	ERR_TIME_NOT_SYNCHRONIZED: The time synchronization provided for use by NDS is not synchronized between the source and target servers. Background modifications to objects will fail when the target server has a modification timestamp higher than the source server.
0xFFFF FD6C	-660	ERR_RECORD_IN_USE: Attempted to purge an NDS database record that is still in use. Action: Run DSRepair.
0xFFFF FD6B	-661	ERR_DS_VOLUME_NOT_MOUNTED: The volume containing the NDS files is not mounted.
0xFFFF FD6A	-662	ERR_DS_VOLUME_IO_FAILURE: Not currently used.
0xFFFF FD69	-663	ERR_DS_LOCKED: Attempted to perform an NDS operation or background process on a server whose NDS database is locked or not open. Action. If the server is unable to open the NDS database and reports an initialization error, run DSRepair.

Hex	Dec	Constant: Description
0xFFFF FD68	-664	ERR_OLD_EPOCH: Attempted to manipulate objects on a server that is using an older epoch of the data. This is a transitory error and NDS synchronization processes will correct it.
0xFFFF FD67	-665	ERR_NEW_EPOCH: Attempted to manipulate objects on a server that is using a newer epoch of the data. This is a transitory error and NDS synchronization processes will correct it.
0xFFFF FD66	-666	ERR_INCOMPATIBLE_DS_VERSION: The version of NDS on the target server is either incompatible with the source server's version or is on the source server's restricted versions list. The following NDS versions are incompatible with all NDS version 5.01 and higher when performing any of the following operations: • An attempt to add a replica to a server running DS.NLM v3.50 or older. • An attempt to assign the Master replica to a server running DS.NLM v4.40 or older when the Master is currently held by a server running DS.NLM v4.63 or higher. • An attempt was made to perform a Move Subtree partition operation when one of the servers involved is running v4.62 or older; or the server with the master replica (which will become the new parent partition) reports one of the server involved is running v0.00.
0xFFFF FD65	-667	ERR_PARTITION_ROOT: Attempted a function that cannot be done on the root partition. For example, the root container of a partition cannot be deleted.
0xFFFF FD64	-668	ERR_ENTRY_NOT_CONTAINER: Not currently used.
0xFFFF FD63	-669	ERR_FAILED_AUTHENTICATION: Passed in a bad password. Bad passwords can occur when the remote IDs on the source or target servers are invalid or one of the servers is using an invalid RSA Public Key. Action: To resolve a suspected remote ID issue, run DSRepair using the advanced menu option "View remote server ID list." Select a server and then select the option to repair remote IDs.
0xFFFF FD62	-670	ERR_INVALID_CONTEXT: An internal NDS error in connection and task management handling. Action: Reload DS.NLM.
0xFFFF FD61	-671	ERR_NO_SUCH_PARENT: Attempted to create or modify an object whose parent object is no longer present or whose specified parent object does not exist.
0xFFFF FD60	-672	ERR_NO_ACCESS: The requester has insufficient rights to perform the operation specified in the request.
0xFFFF FD5F	-673	ERR_REPLICA_NOT_ON: An NDS partition operation or object operation was requested from a server whose replica of the target partition is not in the On state.
0xFFFF FD5E	-674	ERR_INVALID_NAME_SERVICE: Not currently used.
0xFFFF FD5D	-675	ERR_INVALID_TASK: An internal NDS error in connection and task management. Action: Reload DS.NLM.

Hex	Dec	Constant: Description
0xFFFF FD5C	-676	ERR_INVALID_CONN_HANDLE: An internal NDS error in connection and task management. Action: Reload DS.NLM.
0xFFFF FD5B	-677	ERR_INVALID_IDENTITY: An internal NDS error in connection and task management. Action: Reload DS.NLM
0xFFFF FD5A	-678	ERR_DUPLICATE_ACL: Attempted to add an ACL attribute value without overwriting or deleting the existing attribute value that an object already has on the ACL attribute for the specified trustee and privilege.
0xFFFF FD59	-679	ERR_PARTITION_ALREADY_EXISTS: Attempted to create a partition that already exists.
0xFFFF FD58	-680	ERR_TRANSPORT_MODIFIED: An inconsistency occurred when attempting to connect to the target server. The inconsistency is usually caused by faulty LAN drivers, LAN cards, or routers.
0xFFFF FD57	-681	ERR_ALIAS_OF_AN_ALIAS: Attempted to alias an alias
0xFFFF FD56	-682	ERR_AUDITING_FAILED: An internal auditing error occurred.
0xFFFF FD55	-683	ERR_INVALID_API_VERSION: Attempted a request that is not valid or supported by the version of NDS on the server. Action: Update the software.
0xFFFF FD54	-684	ERR_SECURE_NCP_VIOLATION: The connection request failed because either the client or the server does not support NCP packet signatures.
0xFFFF FD53	-685	ERR_MOVE_IN_PROGRESS: A move operation is currently in progress. Before requesting another move, wait until the current move has synchronized to all replicas.
0xFFFF FD52	-686	ERR_NOT_LEAF_PARTITION: A subtree can only be moved if it is a leaf partition.
0xFFFF FD51	-687	ERR_CANNOT_ABORT: Attempted to abort a partition operation when the operation is not in its initial state or when the replica does not indicate that a partition operation is taking place.
0xFFFF FD50	-688	ERR_CACHE_OVERFLOW: Not currently used in NDS v5.95 or higher.
0xFFFF FD4F	-689	ERR_INVALID_SUBORDINATE_COUNT: Indicates an inconsistency in the count of subordinate objects for an object. Action: Run DSRepair.
0xFFFF FD4E	-690	ERR_INVALID_RDN: Indicates an inconsistency regarding the name of an object. Action: Run DSRepair.
0xFFFF FD4D	-691	ERR_MOD_TIME_NOT_CURRENT: Indicates an inconsistency between an attribute's timestamp and the object's modification timestamp. Action: Run DSRepair.

Hex	Dec	Constant: Description
0xFFFF FD4C	-692	ERR_INCORRECT_BASE_CLASS: Indicates an inconsistency regarding an object's base class. Action: Run DSRepair.
0xFFFF FD4B	-693	ERR_MISSING_REFERENCE: Attempted to manipulate an object with a zero creation timestamp. Action: Run DSRepair on the master replica that holds the object.
0xFFFF FD4A	-694	ERR_LOST_ENTRY: NDS cannot update an entry that has not yet been received as a new entry. NDS replica synchronization processes will correct this error.
0xFFFF FD49	-695	ERR_AGENT_ALREADY_REGISTERED: Attempted to load the DS.NLM when another NLM has already loaded and registered as the NDS Agent.
0xFFFF FD48	-696	ERR_DS_LOADER_BUSY: Attempted to unload and reload the DS.NLM but the DS loader is busy.
0xFFFF FD47	-697	ERR_DS_CANNOT_RELOAD: Attempted to unload and reload the DS.NLM, but an another NLM is still loaded which registered an NDS dependency. Dependent NLM such as DSRepair and DSMerge must be unloaded first.
0xFFFF FD46	-698	ERR_REPLICA_IN_SKULK: Indicates that the target server is currently synchronizing with another server in the replica ring. A server can receive updates from only one server at a time. These errors are transitory and NDS automatically resolves them.
0xFFFF FD45	-699	ERR_FATAL: An unrecoverable error has occurred, and the operation cannot be completed. NDS can recover from transitory conditions that cause this error. For continuous occurrences of this error, contact your Authorized Service Center for assistance.
0xFFFF FD44	-700	ERR_OBSOLETE_API: The client made a request that is no longer supported by the server's version of NDS. Action: Obtain an updated version of the program that is compatible with the current version of NDS.
0xFFFF FD43	-701	ERR_SYNCHRONIZATION_DISABLED: Synchronization (replica or schema) has been disabled. Action: Enable synchronization the dstrace command.
0xFFFF FD42	-702	ERR_INVALID_PARAMETER: An invalid parameter was passed to an internal NDS function. Action: Try again. If it fails again, the NDS database may be corrupted. In this case, run DSRepair.
0xFFFF FD41	-703	ERR_DUPLICATE_TEMPLATE: The default ACLs for the specified class definition in the operational schema already exist and are correct. This error occurs during schema initialization when NDS automatically corrects any errors in the operational schema definitions.
0xFFFF FD40	-704	ERR_NO_MASTER_REPLICA: The server is unable to identify the master replica of the partition. Action: If all servers in the replica ring cannot identify the master replica, use DSRepair to assign a new master replica.

Hex	Dec	Constant: Description
0xFFFF FD3F	-705	ERR_DUPLICATE_CONTAINMENT: The containment list for the specified class definition in the operational schema already exists and is correct. This error occurs during schema initialization when NDS automatically corrects any errors in the operational schema definitions.
0xFFFF FD3E	-706	ERR_NOT_SIBLING: Attempted to merge two entries that are not siblings (do not have the same parent container). The client API may have passed a bad parameter.
0xFFFF FD3D	-707	ERR_INVALID_SIGNATURE: Not currently used.
0xFFFF FD3C	-708	ERR_INVALID_RESPONSE: Attempted a request (usually a partition state request or resolve name request) which is not supported by this NDS version.
0xFFFF FD3B	-709	ERR_INSUFFICIENT_SOCKETS: Attempted to open a socket for an NDS server connection but none were available. NDS automatically recovers from this condition by closing the oldest, inactive connections.
0xFFFF FD3A	-710	ERR_DATABASE_READ_FAIL: During security equivalence checking, an object could not be found. Action: If this error occurs more than once, run DSRepair.
0xFFFF FD39	-711	ERR_INVALID_CODE_PAGE: NDS cannot open the Unicode tables. Action: Copy the Unicode tables to the right place.
0xFFFF FD38	-712	ERR_INVALID_ESCAPE_CHAR: Not currently used.
0xFFFF FD37	-713	ERR_INVALID_DELIMITERS: Attempted to pass resolve name a name with invalid delimiters.
0xFFFF FD36	-714	ERR_NOT_IMPLEMENTED: Requested functionality which is not supported on this server or this NDS version.
0xFFFF FD35	-715	ERR_CHECKSUM_FAILURE: In NetWare 4.x, indicates that the received packet failed the checksum verification process. Not used in NetWare 5.0.
0xFFFF FD34	-716	ERR_CHECKSUMMING_NOT_SUPPORTED: The request for checksumming failed because NDS could not set it up.
0xFFFF FD33	-717	ERR_CRC_FAILURE: The CRC check on the reassembled logical packet failed. Action: Check for faulty LAN drivers, LAN cards, and routers.
0xFFFF FD32	-718	ERR_INVALID_ENTRY_HANDLE: Not currently used.
0xFFFF FD31	-719	ERR_INVALID_VALUE_HANDLE: Not currently used.
0xFFFF FD30	-720	ERR_CONNECTION_DENIED: Attempted a connection request when a WANMAN policy prohibits communication with this server during this time period.
0xFFFF FD2F	-721	ERR_NO_SUCH_FEDERATION_LINK: Not used in NetWare 5.0.
0xFFFF FD2E	-722	ERR_OP_SCHEMA_MISMATCH: During initialization or upgrading, NDS attempted to update a class or attribute definition, and the class or attribute is missing from the operational schema.

Hex	Dec	Constant: Description
0xFFFF FD2D	-723	ERR_STREAM_NOT_FOUND: Attempted to open an NDS file, and the file cannot be found. Usually indicates that NDS has not been installed. Can also indicate that an NDS file is corrupted. Action: Depending on the cause, install NDS or run DSRepair.
0xFFFF FD2C	-724	ERR_DCLIENT_UNAVAILABLE: Not used in NetWare 5.0.
0xFFFF FD2B	-725	ERR_MASV_NO_ACCESS: Attempted to access an object that the subject does not have clearance for. This error does not imply that the object exists.
0xFFFF FD2A	-726	ERR_MASV_INVALID_REQUEST: Attempted a request that does not have the correct information to be completed.
0xFFFF FD29	-727	ERR_MASV_FAILURE: Attempted a request with an unsupported feature, the wrong version, or a corrupted label or range.
0xFFFF FD28	-728	ERR_MASV_ALREADY_EXISTS: Attempted a request that has already been completed: (1) to label an already labelled object, (2) to add an authorized range that is already allowed for an object, or (3) to split a container that has a proposed label but also contains children.
0xFFFF FD27	-729	ERR_MASV_NOT_FOUND: Attempted a request with a label that is not defined.
0xFFFF FD26	-730	ERR_MASV_BAD_RANGE: Attempted a request with an invalid range. A range is invalid when (1) the default range is not within the subject's authorized range, (2) the range is not within the range supported by the authentication grade, (3) the request attempts to define a default range which is not within an authorized range, or (4) the request attempts to delete the last authorized range for a subject (a subject must be authorized in at least one range).
0xFFFFFD25	-731	ERR_VALUE_DATA: Unable to read data from the NDS database. Usually indicates that the database is corrupted. Action: Run DSRepair.
0xFFFF FD24	-732	ERR_DATABASE_LOCKED: Attempted to access the NDS database when it is locked. Action: Try again later.
0xFFFF FD21	-735	ERR_NOTHING_TO_ABORT: Attempted to abort a partition operation, but no partition operations are currently in progress.
0xFFFF FD20	-736	ERR_END_OF_STREAM: Attempted to read beyond the end of a stream file. Indicates that all the data has been read.
0xFFFF FD1F	-737	ERR_NO_SUCH_TEMPLATE: Attempted to delete an ACL template which does not exist.
0xFFFF FD1E	-738	ERR_SAS_LOCKED: Not used in NetWare 5.0.
0xFFFF FD1D	-739	ERR_INVALID_SAS_VERSION: Not used in NetWare 5.0.
0xFFFF FD1C	-740	ERR_SAS_ALREADY_REGISTERED: Not used in NetWare 5.0.
0xFFFF FD1B	-741	ERR_NAME_TYPE_NOT_SUPPORTED: Internal NDS error.

Hex	Dec	Constant: Description
0xFFFF FD1A	-742	ERR_WRONG_DS_VERSION: Attempted a resolve name with constraints and this NDS version does not support the requested constraint.
0xFFFF FD19	-743	ERR_INVALID_CONTROL_FUNCTION: Attempted to pass an invalid subverb value to the control verb.
0xFFFF FD18	-744	ERR_INVALID_CONTROL_STATE: Attempted to pass an invalid partition state during a partition operation.
0xFFFF FD17	-745	ERR_CACHE_IN_USE: Attempted to access NDS cache when it is being built. Action: Try again later.
0xFFFF FD16	-746	ERR_ZERO_CREATION_TIME: Attempted to access an object which has a creation time of zero. This is illegal in NetWare 5.x. Action: Run DSRepair.
0xFFFF FD15	-747	ERR_WOULD_BLOCK: Maps to WinSock 2 error, WSAEWOULDBLOCK, which maps to BSD error, EWOULDBLOCK. The requested operation cannot be completed immediately and the socket is marked for nonblocking I/0.
0xFFFF FD14	-748	ERR_CONN_TIMEOUT: Maps to WinSock 2 error, WSAETIMEOUT, which maps to BSD error, ETIMEOUT. The connection timed out. This typically occurs with TCP when IP communications with the peer system do not work. This can be anything from a broken cable to an IP address mismatch.
0xFFFF FD13	-749	ERR_TOO_MANY_REFERRALS: Maps to WinSock 2 error, WSAETOOMANYREFS, which maps to BSD error, ETOOMANYREFS.
0xFFFF FD12	-750	ERR_OPERATION_CANCELLED: Maps to WinSock 2 error, WSAECANCELLED, which maps to BSD error, ECANCELED. The operation has been terminated.
0xFFFF FD11	-751	ERR_UNKNOWN_TARGET: Maps to WinSock 2 error, WSAENETUNREACH, which maps to BSD error, ENETUNREACH. The network cannot be reached from this host. No interface and no route exists with the specified network address.
0xFFFF FD10	-752	ERR_GUID_FAILURE: Attempted to create a GUID for a User object which failed or to get or set the server's GUID which the operating system failed.
0xFFFF FD09	-753	ERR_INCOMPATIBLE_OS: Not used in NetWare 5.0.
0xFFFF FD08	-754	ERR_CALLBACK_CANCEL: Not used in NetWare 5.0
0xFFFF FD07	-755	ERR_INVALID_SYNCHRONIZATION_DATA: Internal NDS error indicating that the obituary data is bad.
0xFFFF FD0C	-756	ERR_STREAM_EXISTS: Internal NDS error indicating that an attempt was made to create a file which already exists. This error causes the filename to be incremented to a unique name.
0xFFFF FD0B	-757	ERR_AUXILIARY_HAS_CONTAINMENT: Attempted to define an auxiliary class with containment rules. Auxiliary classes cannot define containment.
0xFFFF FD0A	-758	ERR_AUXILIARY_NOT_CONTAINER: Attempted to define an auxiliary class as a container class. An class can be either an auxiliary class or a container class, but not both.

Hex	Dec	Constant: Description
0xFFFF FD09	-759	ERR_AUXILIARY_NOT_EFFECTIVE: Attempted to define an auxiliary class as an effective class. An class can be either an auxiliary class or an effective class, but not both
0xFFFF FD08	-760	ERR_AUXILIARY_ON_ALIAS: Attempted to add an auxiliary class to an alias object. Auxiliary classes cannot be added to alias objects.
0xFFFF FD07	-761	ERR_HAVE_SEEN_STATE: During replica synchronization if a replica is currently in a partition operation and receives an update from a replica that is in a lower replica state with a greater timestamp, the replica refuses the update and returns this error.
0xFFFF FD06	-762	ERR_VERB_LOCKED
0xFFFF FD05	-763	ERR_VERB_EXCEEDS_TABLE_LENGTH
0xFFFF FD04	-764	ERR_BOF_HIT: Attempted to take the iterator object to a position beyond the top of the list. The iterator object is currently on the first entry in the list.
0xFFFF FD03	-765	ERR_EOF_HIT: Attempted to take the iterator object beyond the end of the list. The iterator object is currently at the end of the file.
0xFFFF FD02	-766	ERR_INCOMPATIBLE_REPLICA_VER: Not used.
0xFFFF FD01	-767	ERR_QUERY_TIMEOUT: The specified timeout limit was reached before the request could be completed.
0xFFFF FD00	-768	ERR_QUERY_MAX_COUNT: Not used.
0xFFFF FCFF	-769	ERR_DUPLICATE_NAMING: Attempted to modify the naming attributes of a class definition and the class already contains the naming attribute.
0xFFFF FCFE	-770	ERR_NO_TRANS_ACTIVE: Attempted to begin an update operation without a lock, a begin transaction state, or both.
0xFFFFFCFD	-771	ERR_TRANS_ACTIVE: Attempted to begin a transaction when one is already in progress.
0xFFFF FCFC	-772	ERR_ILLEGAL_TRANS_OP: Attempted to an add, modify, or delete update with a lock but without a begin transaction state.
0xFFFF FCFB	-773	ERR_ITERATOR_SYNTAX: Attempted to pass in a search filter with an invalid syntax.
0xFFFF FCFA	-774	ERR_REPAIRING_DIB: Attempted an operation, usually a partition operation, while the DIB is being repaired. Action: Submit the request when DSRepair has finished the DIB repaired.
0xFFFF FCF9	-775	ERR_INVALID_OID_FORMAT: Attempted to read the OID (ASN.1 ID) and the OID is in an invalid format.
0xFFFF FCF8	-776	ERR_DS_AGENT_CLOSING: Attempted to perform an NDS operation, and the DS agent on this server is closing. This error does not cause connection to be torn down; it remains intact.
0xFFFF FCF7	-777	ERR_SPARSE_FILTER_VIOLATION: Attempted to modify an object's attribute that is not stored on the sparse replica.

Hex	Dec	Constant: Description
0xFFFF FCF6	-778	ERR_VPVECTOR_CORRELATION_ERR: Indicates that the two attributes, VpVector and VpvUser, which must be correlated, are out of sync. Action: Use the VR snapin to ConsoleOne to synchronize the attributes.
0xFFFF FCF5	-779	ERR_CANNOT_GO_REMOTE: An internal error. A connection which has been authenticated only for local operations attempted to access remote information. This connection's credentials need to be upgraded to access remote information. The upgrade is done automatically.
0xFFFF FCF4	-779	ERR_CANNOT_GO_REMOTE: An internal error. A connection which has been authenticated only for local operations attempted to access remote information. This connection's credentials need to be upgraded to access remote information. The upgrade is done automatically.
0xFFFF FCF3	-780	REQUEST_NOT_SUPPORTED
0xFFFF FCF2	-781	ERR_ENTRY_NOT_LOCAL
0xFFFF FCF1	-782	ERR_ROOT_UNREACHABLE
0xFFFF FCF0	-783	ERR_VRDIM_NOT_INITIALIZED
0xFFFF FCEF	-784	ERR_WAIT_TIMEOUT
0xFFFF FCE	-785	ERR_DIB_ERROR
0xFFFF FCEE	-786	ERR_DIB_IO_FAILURE
0xFFFF FCED	-787	ERR_ILLEGAL_SCHEMA_ATTRIBUTE
0xFFFF FCEC	-788	ERR_SCHEMA_PARTITION
0xFFFF FCEB	-789	ERR_INVALID_TEMPLATE
0xFFFF FCEA	-790	ERR_OPENING_FILE
0xFFFF FCE9	-791	ERR_DIRECT_OPENING_FILE
0xFFFF FCE8	-792	ERR_CREATING_FILE
0xFFFF FCE7	-793	ERR_DIRECT_CREATING_FILE
0xFFFF FCE6	-794	ERR_READING_FILE
0xFFFF FCE5	-795	ERR_DIRECT_READING_FILE
0xFFFF FCE4	-796	ERR_WRITING_FILE
0xFFFF FCE3	-797	ERR_DIRECT_WRITING_FILE
0xFFFF FCE2	-798	ERR_POSITIONING_IN_FILE
0xFFFF FCE1	-799	ERR_GETTING FILE SIZE
0xFFFF E88F	-6001	ERR_TRUNCATING_FILE
0xFFFF E88E	-6002	ERR_PARSING_FILE_NAME
0xFFFF E88D	-6003	ERR_CLOSING_FILE

Hex	Dec	Constant: Description
0xFFFF E88C	-6004	ERR_GETTING_FILE_INFO
0xFFFF E88B	-6005	ERR_EXPANDING_FILE
0xFFFF E88A	-6006	ERR_GETTING_FREE_BLOCKS
0xFFFF E889	-6007	ERR_CHECKING_FILE_EXISTENCE
0xFFFF E888	-6008	ERR_DELETING_FILE
0xFFFF E887	-6009	ERR_RENAMING_FILE
0xFFFF E886	-6010	ERR_INITIALIZING_IO_SYSTEM
0xFFFF E885	-6011	ERR_FLUSHING_FILE
0xFFFF E884	-6012	ERR_SETTING_UP_FOR_READ
0xFFFF E883	-6013	ERR_SETTING_UP_FOR_WRITE
0xFFFF E882	-6014	ERR_OLD_VIEW
0xFFFF E881	-6015	ERR_SERVER_IN_SKULK
0xFFFF E880	-6016	ERR_RETURNING_PARTIAL_RESULTS
0xFFFF E87F	-6017	ERR_NO_SUCH_SCHEMA
0xFFFF E87E	-6018	ERR_SERIAL_NUM_MISMATCH
0xFFFF E87D	-6019	ERR_BAD_RFL_DB_SERIAL_NUM
0xFFFF E87C	-6020	ERR_BAD_RFL_SERIAL_NUM
0xFFFF E87B	-6021	ERR_INVALID_FILE_SEQUENCE
0xFFFF E87A	-6022	ERR_RFL_TRANS_GAP
0xFFFF E879	-6023	ERR_BAD_RFL_FILE_NUMBER
0xFFFF E878	-6024	ERR_RFL_FILE_NOT_FOUND
0xFFFF E876	-6025	ERR_BACKUP_ACTIVE
0xFFFF E875	-6025	ERR_RFL_DEVICE_FULL
0xFFFF E874	-6026	ERR_UNSUPPORTED_VERSION
0xFFFF E873	-6027	ERR_MUST_WAIT_CHECKPOINT
0xFFFF E872	-6028	ERR_ATTR_MAINT_IN_PROGRESS
0xFFFF E871	-6029	ERR_ABORT_TRANSACTION

Directory Services Trace Utilities

7

The DSTrace utility is used on the server console to monitor how NDS is functioning, to determine whether NDS synchronization processes are completing, and to diagnose NDS errors. Starting with NetWare 5.0, NDS has two DSTrace utilities: an internal version that is part of the DS NLM (SET DSTrace) and an external version, DSTrace NLM:

- The internal version is implemented as a SET command, which can be typed on the server console to enable the NDS debug screen, the display of various messages, the configuration of background process intervals, and the starting of background processes.
- The external version displays and logs messages to the screen and log file. It allows no configuration or starting of background processes.

Commands for the DSTrace utilities for NetWare 5.0 and NDS 8 have been divided into the following functional groups:

- Using the DSTrace NLM
- Using basic SET DSTrace commands
- Starting background processes with SET DSTrace
- Tuning background processes with SET DSTrace
- Enabling DSTrace messages with SET DSTrace

7.1 Using the DSTrace NLM

DSTrace NLM uses the NDS event system to register for messages and has been designed to be easy to use and self documenting. To view the help screen, load the DSTrace NLM and then enter

```
HELP DSTRACE
```

The help screen lists all the command options. The DSTRACE TAGLIST command displays the current configuration and highlights all tags that are turned on.

The messages tags are similar to the SET DSTrace flags, but the list includes some that are not available though the SET command. These include

- LDAP messages
- WAN Traffic Manager messages

To enable a tag, you type DSTRACE and then as many letters as required to make the tag unique (usually two letters). To turn off a tag, type DSTRACE and then -tag. For example, the following command turns on file tracing and the tracing of LDAP messages, and turns off the tracing of schema messages:

```
DSTRACE ON FILE LDAP -SCMA
```

7.2 Using Basic SET DSTrace Commands

The basic commands presented in this section control the general status of the SET DSTrace screen. DSTrace uses the following syntax to initiate the basic functions:

```
SET DSTRACE = <option>
```

Replace option with one of the following.

Table 7-1 Replica Options

Option	Action and Explanation
ON	Enables the NDS debug trace screen. This command restores the old trace bits, if they have been saved, or if no trace vector flags have been previously set, the ON and MIN debug messages are set. This option performs the same functions as the SET parameter, "NDS Trace to Screen = ON."
OFF	Disables the debug trace screen. This command saves the currently set trace vector flags and clears the current trace flags. This option performs the same function as the SET parameter, "NDS Trace to Screen = OFF."
ALL	Enables the debug trace screen and enables all debug trace message flags.

You can save the trace messages to a file for later examination. The default name for the file is SYS:SYSTEM\DSTRACE.DGB. To begin saving screen messages to this file, enter the following command:

```
SET TTF = ON
```

To end saving screen messages and close the file, enter the following command:

```
SET TTF = OFF
```

These commands perform the same functions as the SET parameter "NDS Trace to File = ON/OFF."

7.3 Starting Background Processes with SET DSTrace

The following commands can be used to start background processes. DSTrace uses the following syntax for these commands:

```
SET DSTRACE = *<Flag>
```

Table 7-2 Commands to Start Background Processes

Flag	Action and Explanation
(period)	Instructs NDS to unload and reload.
A	Resets the address cache on source server.
AE	Enables the address cache on source server.
AD	Disables the address cache on source server.
B	Schedules the back link process to begin execution on the source server in one second.

Flag	Action and Explanation
CT	Displays the source server's outbound connection table and the current statistical information for the table. These statistics do not give any information about the inbound connections from other servers or clients to the source server.
CTD	Displays, in comma delimited format, the source server's outbound connection table and the current statistical information for the table. These statistics do not give any information about the inbound connections from other servers or clients to the source server.
D <hex>	Removes the specified local entry ID from the source server's Send All Object list. The entry ID must specify a partition root object that is specific to the server's local database. This command is usually used only when a Send All Updates is endlessly trying and cannot be completed because a server is inaccessible.
E	Re-initializes the source server's entry cache.
F	Schedules flat cleaner process, which is part of the janitor process, to begin execution on the source server in five seconds.
G<hex>	Rebuilds the change cache of the specified root partition ID. New for NDS 8.
H	Schedules the replica synchronization process to begin execution immediately on the source server.
HR	Clears the in-memory last sent vector. New for NDS 8.
I<hex>	Adds the specified local entry ID to the source server's Send All Object list. The entry ID must specify a partition root object that is specific to the server's local database. The replica synchronization process checks the Send All Object list. If the entry ID of a partition's root object is in the list, NDS synchronizes all objects and attributes in the partition, regardless of the value of the Synchronized Up To attribute.
J	Schedules the purge process which is part of the replica synchronization process to begin execution on the source server.
L	Schedules the limber process to begin execution on the source server in five seconds.
M<bytes>	Changes the maximum file size used by the source server's DSTrace file. The command can be used regardless of the state of the debug file. The <bytes> specified must be a hexadecimal value between 10000 bytes and 100MB. If the value specified is higher or lower than the specified range, no change occurs.
P	Displays the current values of the NDS tunable parameters.
R	Resets the DSTrace file to zero bytes.
S	Schedules the NDS replica synchronization process to begin execution on the source server in three seconds. Only replicas already scheduled to be synchronized will be synchronized.

Flag	Action and Explanation
SS	Schedules the schema synchronization process to begin immediately on the source server. Only those target servers which have not been successfully synchronized in the last 24 hours will be synchronized.
SSA	Schedules the schema synchronization process to begin immediately and forces schema synchronization with all target servers, even if they have been synchronized in the last 24 hours.
SSD	Resets the source server's Target Schema Sync list. This list identifies which servers the source server should synchronize with during the schema synchronization process. A server which does not hold any replicas sends a request to be included in the target list of a server that contains a replica with its server object.
SSL	Prints the schema synchronization list of target servers.
ST	Displays the status information for the background processes on the source server.
STX	Displays the status information for back link process (external references) on the source server.
STS	Displays the status information for schema synchronization process on the source server.
STO	Displays the status information for back link process (obituaries) on the source server.
STL	Displays the status information for the limber process on the source server.
U <entryID>	If the command does not include an entry ID, changes the status of any server that has been previously labeled "down" to "up." If the command includes a local entry ID, changes the status of the specified server from "down" to "up." Entry IDs are specific to the source server's database and must refer to an object that represents a server.
Z	Displays the currently scheduled tasks.

7.4 Tuning Background Processes

SET DSTrace allows you to adjust, or tune, several parameters that control restrictions, timeouts, and limits placed on specific NDS activities. The current values of the parameters can be viewed by using the SET DSTRACE = *P command.

The commands that tune parameters cannot be used in conjunction with any other DSTrace commands. They use the following syntax:

```
SET DSTRACE =  !<flag><value>
```

Most, but not all flags, are followed by a value parameter. The value parameter sets the interval the specified process must wait between subsequent runs.

Table 7-3 *Flags for Tuning Background Processes*

Flag	Action and Explanation
B <int>	Sets the interval, in minutes, for the back link process. The default interval is 1500 minutes (25 hours). The range is 2 to 10080 minutes (168 hours).
D <int>	Sets the inbound and outbound synchronization interval to the specified number of minutes. The default interval is 24 minutes. The range is 2 to 10080 minutes (168 hours).
DI <int>	Sets the inbound synchronization interval to the specified number of minutes. The default interval is 24 minutes. The range is 2 to 10080 minutes (168 hours).
DO <int>	Sets the outbound synchronization interval to the specified number of minutes. The default interval is 24 minutes. The range is 2 to 10080 minutes (168 hours).
E	Schedules the inbound and outbound synchronization processes to begin execution.
EI	Schedules the inbound synchronization process to begin execution.
EO	Schedules the outbound synchronization process to begin execution.
F<int>	Sets the interval, in minutes, for the flat cleaner process. The default interval is 240 minutes (4 hours). The range is 2 to 10080 minutes (168 hours).
H<int>	Sets the interval, in minutes, for the heartbeat synchronization process. The default interval is 30 minutes. The range is 2 to 1440 minutes (24 hours).
I<int>	Sets the interval, in minutes, for the heartbeat synchronization process. The default interval is 30 minutes. The range is 2 to 1440 minutes (24 hours).
J<int>	Sets the interval, in minutes, for the janitor process. The default interval is 2 minutes. The range is 1 to 10080 minutes (168 hours).
M	Reports the maximum memory used by NDS.
N [0 | 1]	Sets the name form: zero (0) specifies hex only, and one (1) specifies full dot form. Available only on NDS 8.
SI<int>	Sets the interval, in minutes, for the inbound schema synchronization process. The default interval is 24 minutes. The range is 2 to 10080 minutes (168 hours).
SO<int>	Sets the interval, in minutes, for the outbound schema synchronization process. The default interval is 24 minutes. The range is 2 to 10080 minutes (168 hours).
SI0<int>	Disables the inbound schema synchronization process for the specified number of minutes. The default interval is 24 minutes. The range is 2 to 10080 minutes (168 hours).
SO0<int>	Disables the outbound schema synchronization process for the specified number of minutes. The default interval is 24 minutes. The range is 2 to 10080 minutes (168 hours).
T<int>	Sets the interval, in minutes, for checking the server's UP state. The default interval is 30 minutes. The range is 1 to 720 minutes (12 hours).
V[<ver>[,<ver>]]	Lists the restricted NDS versions. If no versions are listed, there are no restrictions. Each version is separated by a comma.
W<int>	Sets the interval, in ticks, to wait after getting an IPX timeout before resending the packet. The default interval is 20 ticks. The range is 1 through 2000 ticks.

Flag	Action and Explanation
X<int>	Sets the number of IPX retries for the DS client (the NDS-server-to-NDS-server communication). The default number is 3. The range is 1 through 50. When the retry count is exceeded, NDS error -625 is displayed.
Y<int>	Sets the factors used to estimated trip delay. This value is used in the following IPX timeout equation: timeout = (T * Y) = Z. T is equal to the ticks required to get to the destination server; Y is the factor; and Z is an additional delay described in the next row.
Z<int>	Sets an additional delay for the IPX timeout. To increase timeout, change this parameter first. This parameter is used in the following IPX timeout equation: timeout = (T * Y) = Z. T is equal to the ticks required to get to the destination server; Y is the factor; and Z is the additional delay.

7.5 Enabling DSTrace Messages with SET DSTrace

NDS trace messages have been divided into groups by background process, operation, or task. These commands use the following syntax:

```
SET DSTRACE = <operand> <flag> [ <operand> <flag> ]
```

7.5.1 Operands

Multiple operands with flags can be entered in the same command as long as each operand is followed by a message flag.

Operand	Action
+	The plus character enables a DSTrace flag
|	The pipe character enables a DSTrace flag
-	The minus or hyphen character disables a DSTrace flag.
&	The ampersand character disables a DSTrace flag.
( )	The left parenthesis character exclusively enables one DSTrace flag while simultaneously disabling all currently enabled DSTrace flags. The command must end with a right parenthesis, enclosing the enabled DSTrace flag in parentheses. This operand can be used only once in a command and cannot be used in conjunction with the enable and disable operands.

7.5.2 Flags

Flag	Action and Explanation
AUDIT or A	Not currently used in NetWare 5.x and NDS 8.
AUDITNCP or AN	Not currently used in NetWare 5.x and NDS 8.

Flag	Action and Explanation
AUDITSKULK or AS	Not currently used in NetWare 5.x and NDS 8.
AUTHEN	Authentication messages and error reports.
BACKLINK or BLINK	Back link and inbound obituary messages and error reports.
BUFFERS	Inbound and outbound packet buffers that contain data being received in conjunction with, or in response to, an NDS request.
CBUFFERS or CBUF	Client buffer messages.
CHANGE	Change cache messages.
COLLISION or COLL	Status and error reports concerning an object's update information when the update has been previously received.
DEFRAG	See VALUE_DEFRAG.
DRL	Distributed reference link messages.
DSAGENT or DSA	Status and error reports concerning inbound requests from other servers or clients.
EMU	Bindery emulation status messages and error reports concerning the bindery services provided by, or serviced by, the source server.
ERRORS or ERR or E	Not currently used in NetWare 5.x and NDS 8.
ERRET	Not currently used in NetWare 5.x and NDS 8.
FRAGGER or FRAG	Messages from the NCP fragger which breaks NDS messages into NCP-sized messages.
IN	Status messages and error reports concerning inbound requests and processes.
INIT	Status messages and error reports concerning the NDS database initialization process.
INSPECTOR or I	Status messages and error reports concerning the integrity of the objects in the source server's local database. Warning: Use of this flag increases the demands on the source server's disk storage system, memory, and processor. Do not leave this flag enabled unless objects are being corrupted.
JANITOR or J	Status messages and error reports concerning the following background processes: janitor, replica synchronization, and flat cleaner.
LIMBER	Limber status messages and error reports.
LOCKING or LOCKS	Status messages and error reports concerning the use and manipulation of the source server's local database locks.
LOST	Lost entry messages.
MIN	Various messages and errors from different sources in NDS.
MISC	Various messages and errors from different sources in NDS.
MOVE	Messages from the move partition or move subtree operations.
OBIT	Messages from the obituary process. New for NDS 8.

Flag	Action and Explanation
PART	Status messages and error reports about partition operations from background processes and from the request processing.
RECMAN	Status messages and error reports concerning the manipulation of the source server's database.
REPAIR	Status messages pertaining to the interaction of NDS with the DSRepair NLM.
RESNAME or RN	Status messages and error reports concerning the processing of resolve name requests.
SAP	Status messages and error reports concerning the processing SAP (Service Advertising Protocol) information and packets.
SCHEMA	Status messages and error reports concerning the schema synchronization process.
SKULKER or S or SYNC	Status messages and error reports concerning the replica synchronization process
STREAMS	Status messages and error reports concerning the processing of attributes with a Stream syntax.
TH	Status messages and error reports concerning the scheduling of various background processes.
TIMEVECTOR or TV	Various messages pertaining to the following attributes: Synchronize Up To, Replica Up To, and Transitive Vector.
VALUE_DEFRAG or DEFRAG	Not currently used.
VCLIENT or VC	Status messages and error reports concerning the establishment or deletion of connections with other servers.

7.5.3 Sample Commands

To enable the back link process messages, enter one of the following commands:

```
SET DSTRACE = +BLINK
```

or

```
SET DSTRACE= |BACKLINK
```

To disable the back link process messages, enter one of the following commands:

```
SET DSTRACE= -BLINK
```

or

```
SET DSTRACE= |BACKLINK
```

To enable several types of messages, enter the following command:

```
SET DSTRACE = +BLINK +LIMBER +MISC
```

There must be at least once space between each command.

To enable and disable several types of messages in the same command, use the following example as a pattern:

```
 SET DSTRACE= +BLINK  -LIMBER  +MISC  -JANITOR
```

To disable all previously enabled messages and enable one type, for example Miscellaneous messages, enter the following command:

```
SET DSTRACE= (MISC)
```

Revision History

The following table lists all changes made to the NDK: Novell eDirectory Technical Overview documentation:

March 1, 2006	Fixed formatting issues.
October 5, 2005	Transitioned to revised Novell documentation standards.
February 2004	Renamed the product name from “NDS” to “Novell eDirectory” at relevant instances.
November 1999	Added LDAP information to Chapter 1
March 1998	Added NDS 7.x information.