Chapter 4

NDS Services and Supporting Services

Overview

This chapter discusses the different services Novell Directory Services provides and those that support NDS operations. Specifically, the chapter discusses the following concepts:

- Naming services
- Security
- External synchronization
- Time synchronization

Important In this document, an *object* refers to a logical entity in the Directory tree, such as an object representing a user or a printer. In the NDS source code, an object is referred to as an *entry*.s

Concepts to Know

To understand this chapter, you should be familiar with the following Novell Directory Services concepts:

- Partitions
- Replicas
- Entries and objects

Name Services

A name service maps network names to addresses. For example, a server, Ibis, might have the network address 010125DE. A naming service allows the Directory to associate the name Ibis with the server located at network address 010125DE. NDS provides two naming services:

- NDS name service
- Bindery services

NDS Name Service

NDS is an object-oriented, global, information database. An NDS database can handle up to 16 million entries per server. As this number indicates, the large databases typically managed by NDS can be very difficult to administer server by server.

To resolve that difficulty, NDS uses a hierarchical name space rather than a flat name space. This hierarchical organization of objects allows the database to be mapped as a tree, which in turn, allows the database to be partitioned by its subtrees. And because the object names contain the hierarchy information, users can globally access the network resources. The network administrator can also administer the entire tree and its objects from a single point.

NDS Objects and Attributes

NDS consists of objects and attributes based on an extensible schema. Objects are represented on each server by a physical entry. The schema contains the rules for forming these objects and entries and their hierarchy. For information about the NDS schema, see *Novell Directory Services Schema Specification.*

The NDS schema defines two types of objects:

- Container objects
- Leaf objects

Container Objects. Containers are simply objects that can contain other objects. The Country, Locality, Organization, and Organizational Unit objects are container objects.

Leaf Objects. Leaf objects are objects that cannot contain any other objects. These objects are usually the network resources. Users, Printers, and NCP Servers are examples of leaf objects.

Object Attributes

Object class definitions specify both mandatory and optional attribute. Mandatory attributes are those that an object must have values for before that object can exist in the Directory tree. If an object is missing a value for a mandatory attribute, its base class is changed to *Unknown,* since that class specifies no mandatory attributes.

Optional attributes are those that may or may not have values without affecting the object's validity. An object may not have a value for an attribute not defined as mandatory or optional for its object class.

Attributes can be single- or multivalued. A single-valued attributed can have only one value. For example, an object's login script is a single-valued attribute because each object can have only one login script. A multivalued attribute can have more than one value. For example, an object's phone number is a multivalued attribute as each object can have more than one phone number.

Object Names

Every object must have a unique name. This name and its location in the hierarchy form the object's name context. Names lead from the leafmost, or most subordinate, object to the most superior object. As an example, let's use the following name:

.CN=RobF.OU=HQ.O=Novell

In this name, CN=RobF is the name of the most subordinate object while O=Novell is the name of the most superior object. The full name shows the path from CN=RobF to O=Novell

Distinguished and Relative Distinguished Names. A name containing the name of each object in the object's path is called the *Distinguished Name.* The name .CN=RobF.OU=HQ.O=Novell is an example of a Distinguished Name.

A *Relative Distinguished Name* is the leafmost portion of the name. The following are examples of Relative Distinguished Names using the name above:

RobF

CN=RobF

The client appends the object's default context to its Relative Distinguished Name.

Typed and Typeless Names. NDS allows two general types of names:

- *Typed Names.* These names contain the object's location in the hierarchy, as well as the type of each object in the name. For example, .CN=RobF.OU=HQ.O=Novell is a typed name. It defines the name *RobF* as the Common Name of the object, *HQ* as an Organizational Unit object, and *Novell* as an Organization object.
- *Typeless Names.* Typeless names contain the object's location in the hierarchy but do not contain the type of each object. For example, .RobF.HQ.Novell is a typeless name.

Bindery Services

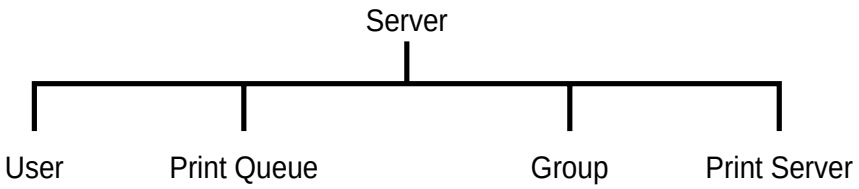
In versions of NetWare before version 4.0, information was not available from a distributed directory. Instead, each server in the network stored a database that contained information such as the name, object ID, and password of every user or object that had access to the services provided by that server. 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.

NDS provides bindery services that allow entries in a container to be accessed both by NDS entries and bindery-based servers and clients.

Bindery Structure

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. A typical bindery might look like this:

Figure 4.1
Example Bindery



Bindery Context

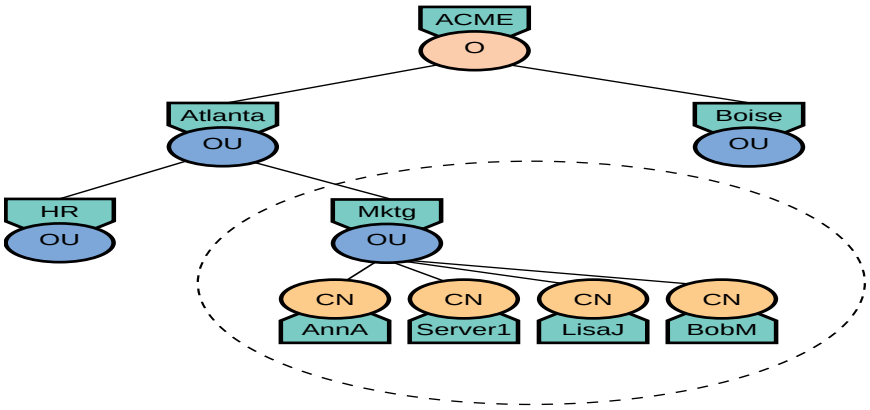
The bindery context is the name of the container object(s) where bindery services is set. For example, in Figure 4.2, the bindery context is set to Mktg.Atlanta.ACME. You can set the bindery context using a SET command at the console.

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 Figure 4.2, 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.

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.

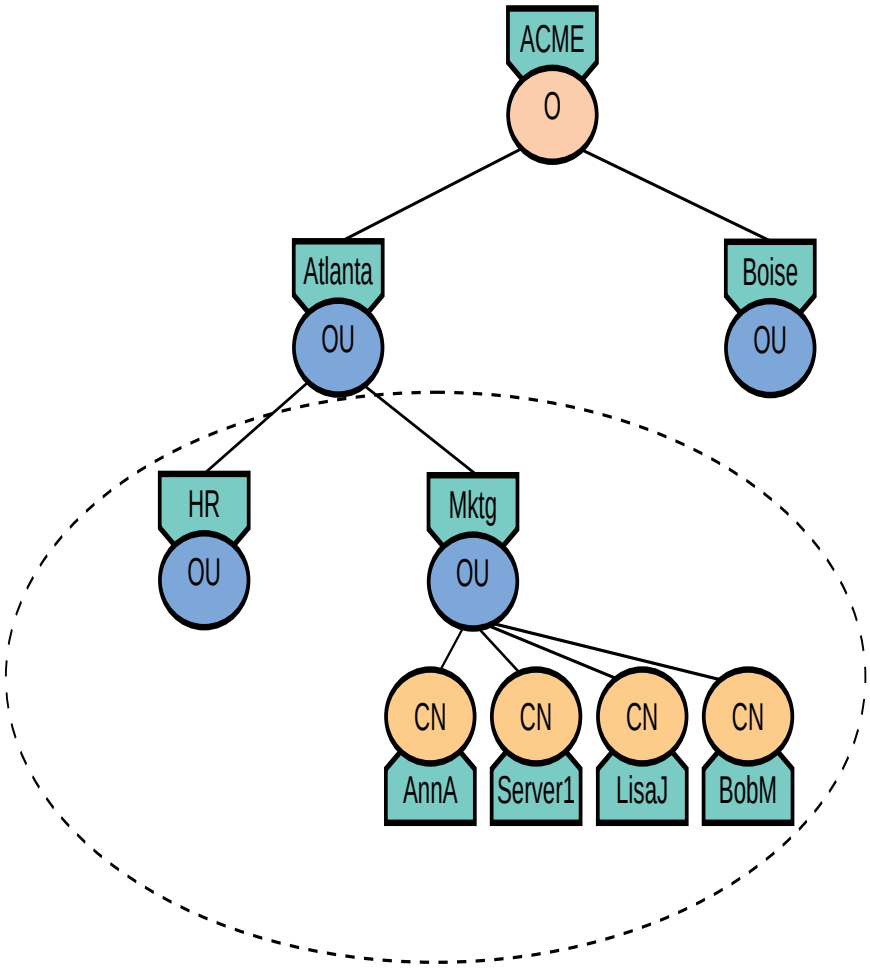
Figure 4.2
Example Bindery



Bindery Context Path

In NetWare versions before NetWare 4.1, you were limited to setting the bindery context at only one container object. Currently, you can set up to sixteen containers as the bindery context. For example, in Figure 4.3, the bindery context could include OU=HR as well as OU=Mktg.

Figure 4.3
Multiple Bindery Contexts



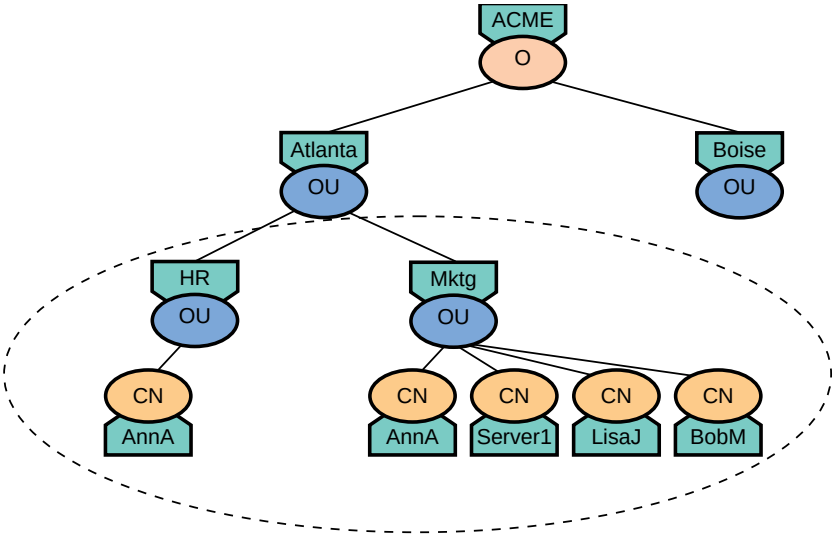
In this case, NDS uses a bindery context path, which is a list of the containers in the bindery context. For example, the bindery context path for a bindery context containing both OU=HR and OU=Mktg in Figure 4.3 would look like this:

HR.Atlanta.ACME
Mktg.Atlanta.ACME

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.

Eclipsing. The Bindery Context Path can create one potential problem. In NDS, two entries can have the same Relative Distinguished Name. For example, in Figure 4.4, user entries with the RDN of “AnnA” exist both in the OU=HR and OU=Mktg containers.

Figure 4.4
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.

Objects in the Bindery

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 ser-

vices maps the entries and attributes stored in NDS to the familiar bindery format, for both objects and properties.

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

NDS bindery services uses dynamic and static bindery objects as follows:

- 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.

- 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.

Bindery Object 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:

Table 4.1
Mutating Bindery Object Types

Bindery Object Type	Corresponding NDS Object Class
1	User
2	Group

Table 4.1
Mutating Bindery Object Types

3	Queue
7	Print Server
309	Profile

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.

Bindery Object 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 Table 4.1.
- 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.

Item Properties and NDS

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 Direc-

tory 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:

1. 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.
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.
3. The server labels the property "undifferentiated" and stores the property's value without attempting to validate the data (see "Undifferentiated Properties" below).

Canonized Properties

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

Table 4.2
Canonized Properties

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
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

Table 4.2
Canonized Properties (Continued)

Bindery Property	NDS Class	NDS Attribute It Maps To
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:

Table 4.3
LOGIN_CONTROL Fields

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).
Time_BitMap	uint8	Login 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.

Table 4.3
LOGIN_CONTROL Fields (Continued)

Field	Type	NDS Attribute It Maps To
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

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

Table 4.4
ACCOUNT_BALANCE Fields (All Object Classes)

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.

ACCOUNT_HOLD. The ACCOUNT_HOLD bindery property maps to NDS properties as follows:

Table 4.5
ACCOUNT_HOLD Field (User)

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.

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

Table 4.6
ACCOUNT_LOCKOUT Fields (Local Server Object Only)

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

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

Table 4.7
DEFAULT_PROFILE Fields (User)

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.

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 4.8
Attribute Bindery 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

Undifferentiated Properties

These 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

NDS Security

Novell Directory Services provides two levels of data security:

- File system security
- NDS access control, which controls the access NDS objects have to other objects and operations

The following sections describe both levels of security.

File System Security

Each NDS object has an associated Entry ID on each server that stores a replica of it. The file system uses this Entry ID as a file system trustee ID to maintain security.

NDS access control rights and file system rights cross over in one respect. If a user has management rights (rights to grant rights) at a File Server entry, that user automatically has Supervisor file rights over all the volumes held on that server.

NDS Access Control

Novell Directory Services uses a process called *access control* to determine the operations Directory entries can perform on other entries and their attributes. Access control restricts many different operations, including creating objects, reading and modifying entry attributes, and comparing attribute values. An access-restricted entry is referred to as a *protected entry.* An entry granted a particular set of access privileges to the protected entry is called the *subject,* or *trustee,* of those privileges.

As a service, access control protects only the entry information stored in the Directory. An application can use the Directory to store access control information specific to the application, but the Directory does not manage, evaluate, or apply this data on the application's behalf.

Because NDS defines and enforces Directory access, a Directory client application cannot access the Directory without using NDS access control. If a Directory client application must use access control in its session with clients, NDS can centrally store and retrieve access control information.

By default, an entry has the access privileges of:

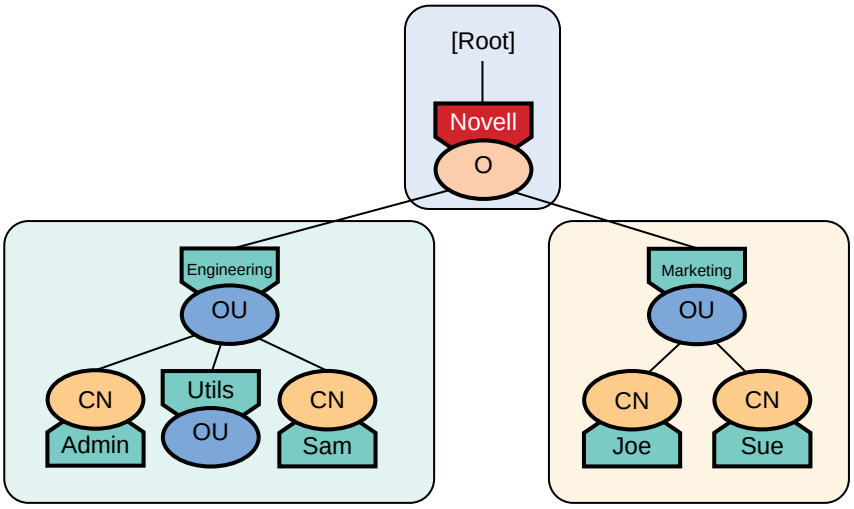
- Every entry in its Distinguished Name, unless one of these objects has an inherited rights filter.
- Every special entry name that applies ([Public], [Root], [Creator], and [Self]).
- Every entry in its Security Equals attribute, if its object class definition has that attribute.

For example, in the following illustration, the object Joe.Marketing.Novell would have the same rights as:

- OU=Marketing and O=Novell, because they are in Joe's Distinguished Name (unless an Inherited Rights Filter applies).
- [Root] and [Public].

Sue, if Joe is security equivalent to Sue.

Figure 4.5
Default Rights



Access Control List (ACL)

The Access Control List (ACL) is the key component in determining Directory access control. The ACL determines which operations a trustee entry can do on another entry and its attributes. In the Directory schema, the ACL is a multivalued attribute type assigned as an optional attribute to the object class *Top*. Because all object classes inherit the characteristics of Top, all entries could have an ACL attribute.

The ACL attribute contains three values:

Table 4.9
Access Control List

Field	Type	Description
Trustee ID	uint32	This field contains an Entry ID for a specific entry in the Directory. However, it could also contain a special entry reference such as [Inherited Rights Filter], [Public], [Root], [Creator], or [Self]. An ACL with [Inherited Rights Filter] as the trustee masks or filters privileges granted to an entry.

Table 4.9
Access Control List (Continued)

Field	Type	Description
Protected Attribute ID	uint32	This field contains a reference to the attribute that the *Privilege Set* field (see below) applies to. It could also contain an identifier such as [Entry Rights] or [All Attributes Rights]. If this field contains [Entry Rights], the access privileges apply to the entry that holds this ACL attribute (see "Rights" in this chapter for an explanation of access control rights).
Privilege Set	uint32	This field lists the privileges that have been granted to the subject. If the ACL subject name is [Inherited Rights Filter], the *Privilege Set* field lists the rights that can be granted on that entry, although they might not have been granted.

The ACL is an attribute on the object that is being accessed. The ACL attribute lists the trustees and the rights they have to the object. For example, if object Joe had Browse [Entry Rights] over OU=Engineering, the ACL on object OU=Engineering would look like this:

Table 4.10
Sample ACL

ACL Field	Value
Subject Name (Trustee Name)	CN=Joe.OU=Marketing.O=Novell
Protected Attribute ID	[Entry Rights]
Privilege Set	Browse

ACLs are represented differently to clients and the DS.NLM. The ACL fields have the following representation:

Table 4.11
ACL Fields

ACL Field	Client Representation	DS.NLM Representation
Subject Name (Trustee Name)	Unicode Distinguished Name, including [Public], [Root], and [Inherited Rights Filter]	Local Entry ID, including ID_PUBLIC or ID_INHERITED_ RIGHTS_FILTER
Protected Attribute ID	Unicode attribute name or [Entry Rights] or [All Attributes Rights]	Local Schema ID including ID_ENTRY_RIGHTS, ID_ALL_ATTR_ RIGHTS
Privilege Set	Entry bit set or attribute bit set	

Default ACL Templates. In the NDS schema, most object classes specify a default ACL template that is used to create an ACL for a new entry.

This default template provides basic access control for the new entry, allowing it to function in the Directory. Different object classes have different default ACLs to reflect their different needs. For example, the User object class's default ACL grants that User entry the *Write* right to its Login Script attribute. This allows users to change their login scripts as necessary.

Rights

The *Protected Attribute ID* field can identify three types of access control rights:

- [Entry Rights]
- [All Attributes Rights]
- Specific attribute rights

These rights are granted to the entry's trustee.

[Entry Rights]. [Entry Rights] give a trustee rights that affect an entry as a whole, not just its attributes. The following are [Entry Rights]:

- *Browse [B]* This right allows a trustee to discover Directory entries.
- *Create [C]* This right allows a trustee to create child entries (new entries that are subordinate to the entry in the tree).
- *Delete [D]* This right allows a trustee to delete an entry. The *Delete* right does not allow a trustee to delete a container entry that has subordinate entries.
- *Rename [R]* This right allows a trustee to rename an entry.
- *Supervisor [S]* This right gives a trustee all rights to an entry and its attributes.

Attribute Rights. A trustee can have rights to all an entry's attributes or to a specific attribute on that entry. For example, User entries may have the *Write* attribute right to their Telephone Number attribute. This allows a user to change his or her Telephone Number attribute as necessary. Granting that same User *Write* [All Attributes Rights] on its attributes would allow that user to modify all writable attributes.

The following rights can be granted to all an entry's attributes or to specific attributes of an entry:

- *Compare [C]* This right allows a trustee to compare, or see if an attribute contains a given value.
- *Read [R]* This right allows a trustee to read an attribute value. The *Read* right confers the *Compare* right.

Note The schema defines some Directory attributes as "hidden." Trustees cannot read such an attribute's value even if they have the *Read* right. For information about hidden attributes, see *Novell Directory Services Schema Specification.*s

- *Write [W]* This right allows a trustee to add, delete, or modify an attribute value. The *Write* right also gives the trustee the *Add or Delete Self* right to the attribute.
- *Add or Delete Self [A]* Some attributes take entry names as their values. This right allows a trustee to add or delete its name as an attribute value.
- *Supervisor [S]* This right gives a trustee all of the attribute rights.

Note The schema defines some read-only and public-read attributes, such as a Back Link, that trustees cannot modify. For information about read-only and public-read attributes, see *Novell Directory Services Schema Specification.*s

Rights Needed for NDS Operations. As mentioned, Access Control Lists provide security by specifying which NDS entries, or trustees, can perform operations on other NDS entries. The following table explains which rights are required to perform different NDS operations:

Table 4.12
Rights Needed for NDS Operations

Directory Operation	Entry Rights Required	Attribute Rights Required
Compare an attribute value		*Compare* on the specific attribute
Read attribute		*Read* on the specific attribute
List subordinates	*Browse* on child	
Search	*Browse* on base entry *Browse* on entry to be returned	*Compare* on each attribute evaluated in the search filter. *Read* on each attribute returned
Modify entry (add attribute)		*Write* on the attributes to be added
Modify entry (add attribute value)		*Write* on the attributes to be modified
Modify entry (delete attribute)		*Write* on the attributes to be deleted

Table 4.12
Rights Needed for NDS Operations

Modify entry (delete attribute value)		*Write* on the attributes to be modified
Add entry	*Add*	
Delete entry	*Delete*	
Modify RDN	*Rename*	
Add/Delete self from group		*Add or Delete Self* on the Membership attribute

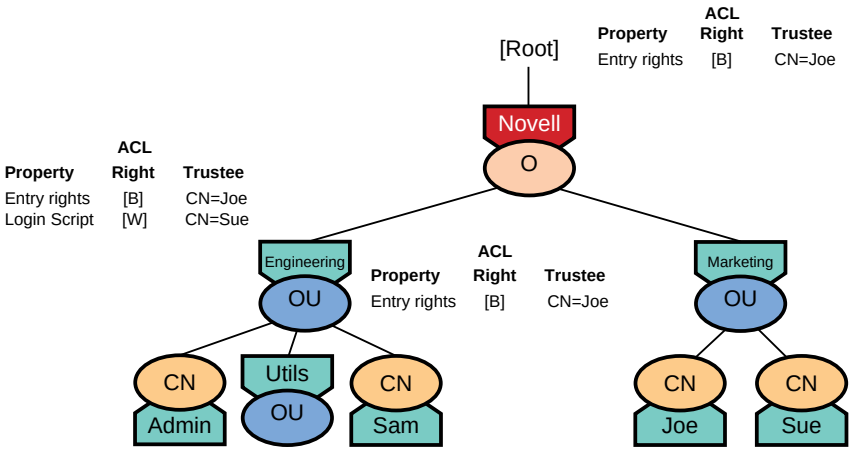
Inheritance

NDS rights "inherit," or flow down the tree the same way NetWare file system rights do: rights granted at a container apply to all subordinate entries within the container and subsequent subordinate containers.

Note Inheritance applies only to [Entry Rights] ACLs and [All Attributes Rights] ACLs. Specific attribute rights are not inherited. An Inherited Rights Filter can prevent rights from being inherited.

For example, in Figure 4.6, Sue and Joe have *Browse* [Entry Rights] to [Root] by default. Through inheritance, Sue and Joe also have *Browse* [Entry Rights] to OU=Engineering and to all of its subordinate entries.

Figure 4.6
Inherited Rights



However, specific attributes rights do not flow down the tree. In the example, if Joe has the *Write* attribute right to OU=Engineering's Login Script, Joe would not automatically have the same right on the entries below OU=Engineering.

Note An ACL can grant rights to an attribute that is not present on the entry. This allows administrators to grant rights, through inheritance, on all the entries under a container, even if that container does not hold the same attributes as the container. For example, a container may not have a Telephone Number attribute, but granting the container the *Read* [All

Attributes Rights] on its entry allows all the users in the container to read all their attributes, including the Telephone Number attribute.

NDS calculates an entry's inherited rights by checking the entry's ACL attribute and then moving up the tree until it finds an Inherited ACL attribute (on the partition root entry). On each ACL it finds, it checks for [Inherited Rights Filters]. All rights not masked by an [Inherited Rights Filter] flow down the tree.

In the example tree above, NDS would calculate Joe.Marketing.Novell's inherited rights by walking up the tree to Marketing.Novell, which as a partition root entry, has an Inherited ACL. NDS would check the Inherited ACL for rights Joe receives by inheritance.

Security Equivalence

Security equivalence simply means that one entry has the same rights as another entry in the Directory tree. Before the system can calculate an entry's rights, it must determine its security equivalences. Every entry in the Directory tree is security equivalent to the following entries:

- [Public], even if the entry is not authenticated
- The [Root] entry of the tree, if authenticated
- Every entry contained in its Distinguished Name
- Every entry listed in its Security Equals attribute

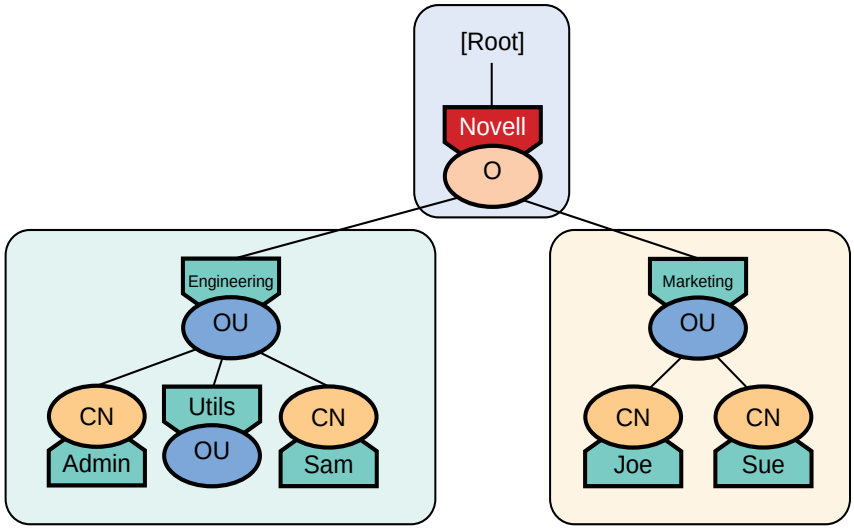
Creating the Security Equivalence Vector. The Security Equivalence Vector (SEV) for a Directory entry is an array that holds the following information for each entry that is security equivalent:

- Trustee ID
- Groups to which the trustee belongs, including any subtrees to which the subject belongs (For example, Testing.Engineering.Novell belongs to the subtrees Engineering and Novell).
- Public ID

Determining Security Equivalence. When an entry authenticates to the Directory, NDS builds a Security Equivalence Vector and stores it in the connection table.

As an example of how rights are computed, assume that in Figure 4.7 Joe has *Browse* [Entry Rights] to Sam. If Sue is security equivalent to Joe, Sue also has *Browse* [Entry Rights] to Sam.

Figure 4.7
Security Equivalence



Now suppose that Sam's ACL does not grant Sue any [Entry Rights] to Sam. To compute Sue's rights, NDS starts at O=Novell and walks down the tree. As it walks down the tree, NDS looks for ACLs on the entries it finds. It finds that Sam's ACL attribute does not give Sue any [Entry Rights] to Sam; however, it also finds that Sue is security equivalent to Joe, which has *Browse* [Entry rights] to Sam. So, Sue gets *Browse* [Entry Rights] to Sam through security equivalence, not through inheritance.

Effective Rights

Effective rights are the sum of all the rights a user has received to Directory entries. A client can list the effective rights a given user, or trustee, has over NDS entries. These rights are received from the following sources:

- Each entry's ACL
- The trustee's security equivalences
- Access privileges inherited from the entry's parents
- Access privileges of the designated subject, [Public]

The algorithm for getting a user's effective rights proceeds as follows:.

Table 4.13
Obtaining a User's Effective Rights

Module	Action
Server holding trustee	1. Sends a *Get Effective Privileges* request to the server.
Responding server	2. Walks the tree, reading the ACL attributes of the entries it finds. 3. Summarizes the trustee's rights and returns them in a *Get Effective Privileges* reply.

Inherited Rights Filters

NDS allows administrators to control how rights are inherited. A special kind of ACL called an *Inherited Rights Filter* (IRF) specifies which rights can be inherited from an object to entries below it in the Directory. The IRF (sometimes called an *inheritance mask*) takes the place of the trustee name in the ACL ("[Inherited Rights Filter]" appears in the Trustee ID field). It can be applied to [Entry Rights], [All Attributes Rights], or specific attribute rights.

Figure 4.8 shows the same tree shown above in Figure 4.7. This time, however, the following ACLs apply:

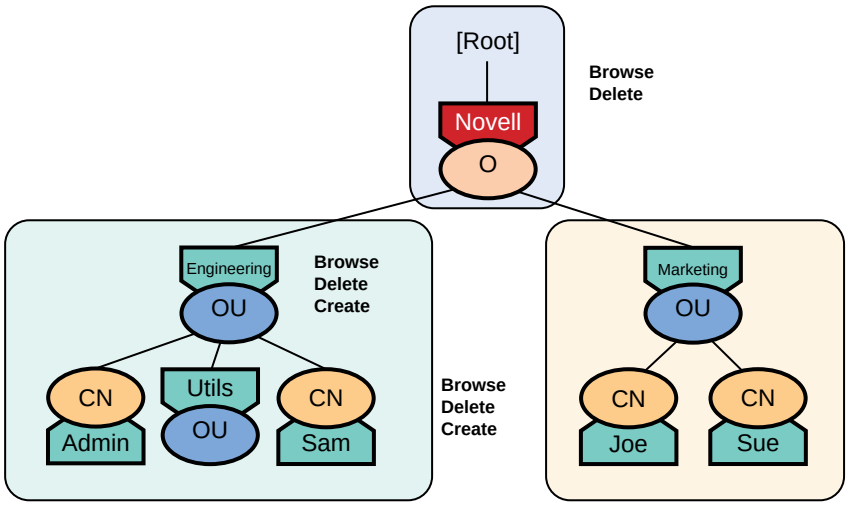
Table 4.14
Example ACLs

Object Holding ACL	Trustee	Protected Attribute	Privilege Set
Novell	Marketing	[Entry Rights]	Browse, Delete
Engineering	Joe	[Entry Rights]	Create
Sam	Sue	[Entry Rights]	Delete

If no object in the Directory has an Inherited Rights Filter, Sue would have the following effective rights:

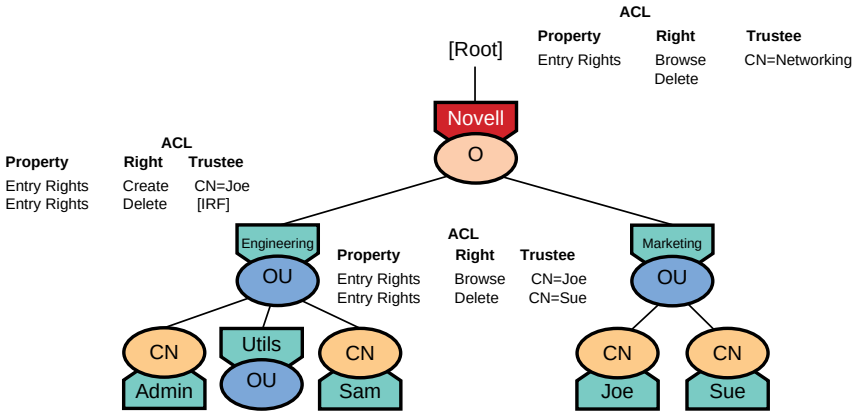
- The *Delete* right to Sam because Sam's ACL grants this right to Sue
- The *Delete* right to Novell and Engineering.Novell because Marketing is part of Sue's Distinguished Name (this makes Sue security equivalent to Marketing)
- The *Create* right to Engineering and Sam because Sue is Security equivalent to Joe
- The *Browse* right to Sam, Engineering, and Novell because Sue is security equivalent to Marketing

Figure 4.8
Sue's Effective Rights



Suppose, however, that Engineering.Novell has an Inherited Rights Filter that masks the *Delete* right.As Figure 4.9 shows, Sue no longer has the *Delete* right to Engineering but retains others.

Figure 4.9
Sue's Rights with an Inherited Rights Filter



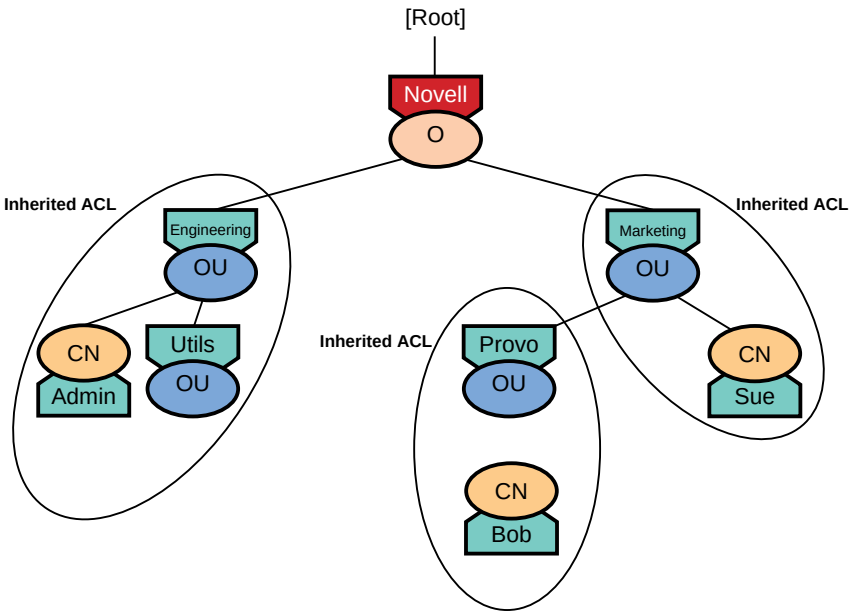
Inherited ACL

An Inherited ACL is an attribute of each partition root entry. This attribute summarizes the effective rights that trustees inherit in order to perform operations on entries. The Inherited ACL attribute is valid only on a partition root entry. Unless an Inherited Rights Filter is in use, these effective rights are granted to all entries in the partition. Having this attribute at the partition root level allows NDS to calculate effective rights for each entry in the partition without walking that entire portion of the Directory tree.

Note Inherited ACLs are not visible to users or clients.

When a client creates a new partition, NDS creates an Inherited ACL as a multivalued attribute attached to the partition's root entry. NDS updates these ACLs when the Directory tree changes. In Figure 4.10 Engineering, Marketing, and Provo have Inherited ACL attributes because they are partition root entries.

Figure 4.10
Inherited ACLs



The name server creates the Inherited ACLs using the basic Security Equivalence Vector (SEV) structure. The algorithm proceeds as follows:

1. The name server walks the tree from the [Root] partition down to the new partition root and gathers inheritable ACLs. When it encounters another ACL containing the same trustee, it replaces the previous (superior) ACL. If the server finds an Inherited Rights Filter, it applies it to all ACLs. It then adds these ACLs to the partition root as inheritable values.

2. The server creates these entries in the partition root's Inherited ACL attribute. It marks them as Inherited ACL attributes by setting the Trustee ID field to a special value.

The Janitor process maintains inherited ACLs. When a client or server modifies ACLs, the Janitor process modifies the ACL for any replicas on a given server that are affected by the change. This new inherited ACL is forwarded only to child partition root entries. Synchronization propagates inherited ACL information among replicas of a partition.

Management Rights

Management rights are the rights that allow a trustee to grant rights to other entries. A trustee has management rights when it has the *Write* attribute right to the entry's ACL attribute. The following rights can grant management rights to a trustee:

- *Write* or *Supervisor* right on the ACL attribute
- *Write* or *Supervisor* [All Attributes Rights]
- *Supervisor* [Entry Rights]

Management rights are important because a trustee that can modify an entry's ACL can control what other entries can do to that entry. The following NDS operations require management rights to complete:

- To modify the schema, a trustee must have management rights to the Directory's root partition.
- Partition operations require the trustee to have management rights on the target partition.
- Backup requires the trustee to have managed rights on the entry being backed up.
- Adding an entry to a write-managed attribute on another entry requires management rights to the entry being added.

Write-Managed Attributes

Write-managed attributes are those attributes whose values are the Distinguished Names of entries in the Directory tree. To add or delete an entry name to a write-managed attribute, a trustee must have management rights to the entry being added; however, the trustee does not need rights to the object being modified. The schema defines write-managed attributes, which include:

- Security Equals
- Group Membership
- Higher Privileges (not used)
- Profile Membership

Bindery Services Security

All bindery objects and properties have an associated security value consisting of two nibbles (4-bit values), one specifying Read rights and the

other Write rights. Table 4.15 shows the possible values a bindery object or property can have.

Table 4.15
Bindery Security Values

Value	Permission
0	ANYONE. Permission granted to all unauthenticated users.
1	LOGGED. Permission granted to all authenticated users.
2	OBJECT. Permission granted only to user logged in as a specified object.
3	SUPERVISOR. Permission granted only to the object's supervisor.
4	BINDERY. Permission granted only through system code (no NCP access).

The bindery stored these values in a single byte and interpreted them in Hex; thus, security value 31h allows only supervisors the Write right to the object, but any logged in user can read it.

NDS emulates access control for bindery calls by adding Access Control Lists (ACLs) to the bindery object, depending on the bindery security value. Table 4.16 shows the NDS access control applied to bindery objects.

Table 4.16
Applying Access Control to Bindery Objects

Value	Permission	Access Control
0	ANYONE	ACL for [Public]
1	LOGGED	ACL for [Root]
2	OBJECT	ACL for the local entry
3	SUPERVISOR	Management rights over the object
4	BINDERY	Not directly supported, but the absence of any other rights implies this value

NDS applies its access control rights to bindery objects, attribute bindery properties, and some canonized bindery properties. Most canonized properties do not allow their security to be modified, and the security value is hard coded in the canonized properties table. Undifferentiated (bag) properties have a fixed field to maintain the bindery-style security for each value of the Bindery Property attribute.

External Synchronization

Novell Directory Services stores information about entries in its own global database. Other, external databases may need to access the information stored in an NDS database. NDS's external synchronization functions allow external databases to synchronize with an NDS Directory.

Currently, external synchronization for NDS consists of NDS Event Services.

Event Services

NDS Event Services provides a way to monitor the activity of NDS on an individual server. When specified NDS events occur, a monitoring NLM can view them and determine what action to take. Event Services can notify the NLM during or after the event. Using Event Services, an NLM application can register one or more functions to be called when specific events occur. These events stay registered until the NLM application specifically unregisters them.

NDS Event Services consists of registration functions and helper functions.

Registration Functions

These allow an NLM application to register and unregister a function to be called when a specific event occurs. Registration functions include the following:

- *DSERegisterForEvent.* This function registers a function to be called when a specific NDS event occurs.
- *DSEUnRegisterForEvent.* This function "unregisters" a function so it will not be called when an NDS event occurs.

Helper Functions . These help access and evaluate the data for the event. If successful, these functions return zero. If unsuccessful, they return a negative value that identifies the error.

- *DDSConvertEntryName.* This converts the object names returned in the *DSEEntryInfo* structure to the requested form.
- *DDSGetLocalAttributeID.* This retrieves the local ID of a specified schema attribute.
- *DDSGetLocalClassID.* This retrieves the local ID for the specified object class.
- *DDSGetLocalEntryID.* This retrieves the local ID for the specified Directory object.
- *DDSGetLocalEntryName.* This retrieves the name of the Directory object associated with the supplied local ID.

Priority. This parameter specifies when the events will be reported. A client can specify one of three possible priorities:

- EP_INLINE. This is synchronous pre-event reporting. It allows the callback to determine if the event should succeed. If the callback

returns a nonzero value, the event is aborted and the callback's return value is returned to the client.

With this priority, the client waits for a response while the callback processes.

When the event is being reported, the callback cannot make any DDS calls because the local database is locked. In addition, the only function it can use from NDS Event Services is *DDSConvertEntryName.*

The callback can sleep (normally only to allocate memory).

This priority faces the most difficult issues when it comes to chained event handlers. You cannot assume that an NDS event will complete if your callback returns zero because the next callback could abort the transaction. To verify that a change has actually occurred, you must register a callback for the EP_JOURNAL or EP_WORK priorities.

- EP_JOURNAL. This is synchronous postevent reporting. The event information is kept in a journal queue that records the events in the order they have occurred.

  A single thread services all the callbacks for the events, so the callback's execution time should be kept as short as possible. (The callback can determine if data should be used. If so, the callback can store the data on a list that another thread will process.)

  If multiple callbacks are registered for the same event, these callbacks must wait until the current callback is processed before the next callback is called.

  This callback can sleep and can use any of the DS and DDS functions.

- EP_WORK. This is asynchronous postevent reporting. The events are reported after they occur but not necessarily in the order that they occurred. They are reported after all of the event's callbacks that are registered for the EP_JOURNAL priority have completed.

  Each callback is run on a separate thread. This frees the event handler from the time constraints of the other two priorities.

  This callback can sleep and can use any of the DS and DDS functions.

Type. This parameter specifies the type of event to associate the callback with. Currently the following #define values can be specified:

Table 4.17
Event Types

	Event	Description
1	DSE_CREATE_ENTRY	Creating a new entry
2	DSE_DELETE_ENTRY	Deleting an existing entry.
3	DSE_RENAME_ENTRY	Renaming an entry.

Table 4.17
Event Types (Continued)

	Event	Description
4	DSE_MOVE_SOURCE_ENTRY	In a move operation, deleting the entry from its original location in the Directory tree.
5	DSE_ADD_VALUE	Adding a value to an entry's attribute.
6	DSE_DELETE_VALUE	Deleting a value from an entry's attribute.
7	DSE_CLOSE_STREAM	Closing a Stream attribute.
8	DSE_DELETE ATTRIBUTE	Deleting an attribute.
9	DSE_SET_BINDERY_CONTEXT	Setting the bindery context on the server.
10	DSE_CREATE_BINDERY_ OBJECT	Creating a bindery object.
11	DSE_DELETE_BINDERY_ OBJECT	Deleting a bindery object.
14	DSE_MOVE_DEST_ENTRY	In a move operation, the new location of the object in the Directory. This event generates the DSE_ADD-VALUE events for all the values.
15	DSE_UNUSED_EXTREF	Deleting an unused external reference.
16	DSE_TRACE	Occurrence of a DSTrace event.
17	DSE_REMOTE_SERVER DOWN	
18	DSE_NCP_RETRY_EXPENDED	
19	DSE_REMOTE_CONN_ CLEARED	

Handler. This is a pointer to a function that the Event Handler will call when the event occurs. These functions retrieve the data related to the event. DSHandler can register separate functions for each event, or the same function for multiple events.

Filtering Event Data

When *DSEventHandler* calls a callback, the callback must determine if the data (pointed to by the *data* parameter) contains information that the NLM application wants to use. For example, if the NLM application cares only about changes to telephone numbers, the callback would use its data only if it contained information for a *Telephone Number* attribute. If the data does not contain *Telephone Number* attribute information, the callback may return no information or an error code, depending on what the NLM application requests.

Filtering by Local ID

In most cases, you can filter using local IDs because data structures in the callback's *data* parameter use IDs rather than names. For example, the *DSEValueInfo* structure contains the following IDs:

perpetratorID
entryID
attrID
syntaxID
classID

While the object names in the Directory are global, the local IDs for entries on individual servers are not. Each object on a server is identified by a local ID that is relevant only on that server. The object's local ID on another server may or may not be the same. Local IDs are also used to identify attributes and object classes.

These local IDs are used to improve reporting speed. Comparing two 32-bit IDs is faster than comparing two strings.

Time Synchronization

Because NDS is a distributed, replicated database, the same object can be modified from different replicas in the network. Time synchronization ensures that multiple changes to the database are done in the order they occur.

Time synchronization ensures that all servers in a Directory tree report a consistent Universal Time Coordinated (UTC) time. NDS needs this consistent time to establish the order of different operations done on different replicas in the Directory tree. Time synchronization may not provide the correct time of day exactly, but it ensures that the servers in the tree remain synchronized.

With time synchronization, NDS can stamp Directory events with a unique *time stamp*, which identifies an event and associates it with a time. Time synchronization makes sure that all time stamps are based on the same time across the network.

Determining the Correct Time

To calculate the correct time, two types of time information are maintained on every server in the Directory tree: time zone information and daylight savings time information.

Time Zone Information

Time zone information is maintained by a *time zone string* on each server in the Directory tree. This string holds three important pieces of information:

- The abbreviation for the local time zone name
- The offset of the local time zone from UTC
- The abbreviation for the local time zone to be used during daylight savings time

The format for displaying this information is as follows:
Normal Time Zone, Offset, Daylight Savings Time Zone

The normal time zone abbreviation indicates the three-letter abbreviation for the time zone in which the server resides.

The offset indicates the number of hours that must be added to the clock to equal UTC. Since UTC is based on the time in Greenwich, England, time zones to the west, including those in the United States, must add hours to equal UTC. These time zones have a positive offset. Time zones to the east of UTC must subtract hours to equal UTC; they have a negative offset.

The daylight time zone abbreviation indicates the standard three-letter abbreviation for the time zone when daylight savings time is in effect.

The format for representing these values is illustrated in the following example, which shows the string needed for a server in Los Angeles, California:
PST8PDT

This string shows that the normal time zone is Pacific Standard Time, the offset is eight hours from UTC, and the daylight savings time zone is Pacific Daylight Time.

Daylight Savings Time SET Parameters

Daylight savings time information is controlled by the following SET parameters:

Table 4.18
Daylight Savings Time SET Parameters

SET Parameter	Default	Explanation
SET Daylight Savings Time Offset=	+1:0:0	An offset, in seconds, *added* to local time at the beginning of daylight savings time. A change causes UTC to be recalculated from local time.
SET Start of Daylight Savings Time= SET End of Daylight Savings Time=	None	Specifies dates or rules for dates when daylight savings time starts or ends.

Table 4.18
Daylight Savings Time SET Parameters (Continued)

SET Parameter	Default	Explanation
SET Daylight Savings Time Status=	OFF	The status may be changed at any time. Doing so does not change the local time, but it does cause UTC to be recalculated.
SET New Time with Daylight Savings Time Status	OFF	This command not only changes the daylight savings time status but also adjusts the local time by the daylight savings time offset. This effectively leaves UTC unchanged.

Daylight savings time status should be set using the "SET Start of Daylight Savings Time=" and "SET End of Daylight Savings Time=" commands. These commands are placed in the AUTOEXEC.NCF file during system initialization. The network administrator can modify them using a text editor.

Calculating UTC Time from Local Time

Whenever time is changed on a server, UTC is recalculated from the current local time. The relationship between local time and UTC is as follows:

UTC = local time+time zone offset-daylight savings time offset

The daylight savings time offset applies only when daylight savings time is in effect.

Synchronizing a Server's Time

Time is synchronized by speeding up or slowing down the *apparent* tick rate of the hardware clock on each server. UTC time is maintained as a software clock on each server that is updated during each hardware clock interrupt.

A NetWare Loadable Module called TIMESYNC.NLM contains all of the functionality required to synchronize time.

Types of Time Servers

Servers or workstations controlled by time synchronization fall into four categories:

- Secondary servers
- Primary servers
- Reference servers
- Single Reference servers

Each type of time server is explained in the following section. All of these servers do three things:

- Provide UTC time to NLMs or client workstations requesting it
- Provide status information indicating whether the UTC time is synchronized
- Adjust their internal clock rates to maintain UTC synchronization

Primary, Reference, and Single Reference servers are considered *time sources.* These servers determine the official network time. During each polling loop, time sources use SAP to advertise their presence and contact all other time sources; then they determine the correct time. Each has a different method of determining correct time, as explained below. In contrast, Secondary servers contact only one server, their time source.

When a server's internal UTC clock is within its *synchronization radius* (or the interval between the server's time and the time sources' time beyond which a server is out of synchronization), the server sets its *synchronization flag* to *true.* This flag indicates that the server's UTC time is synchronized with other servers and can be exchanged as a time stamp with other synchronized servers.

Secondary Servers

Secondary servers are not time sources but try to synchronize with one other server, usually a time source. Either the administrator can choose the time source a Secondary server synchronizes with, or Service Advertising Protocol (SAP) determines it. NetWare 4 servers are Secondary servers by default, except for the first server in a tree.

When a Secondary server is not synchronized with its time source, it tries to adjust its time to correct the error during the next polling interval. The server sets its synchronization flag to *true* when the time discrepancy is less than the synchronization radius and its time source is synchronized.

Primary Servers

Primary servers are time sources that provide the following network services:

- Poll other time sources to determine the correct network time and adjust if necessary
- Provide an official time so that Secondary servers can get the time from a local source
- Provide fault tolerance so that if one time source fails, servers can get the time from another source

After polling all time sources, Primary servers determine if their time matches the network time. If the times do not match, Primary servers adjust their internal time by one-half the error during the next polling

period. If the error is less than the synchronization radius, Primary servers set their synchronization flags to *true*, whether or not other servers are synchronized.

Synchronizing by one-half the discrepancy enables servers to converge. If two or more Primary servers adjusted to the full discrepancy, they could constantly change back and forth but never converge to the correct network time.

Reference Servers

A Reference server is a Primary server that polls other time sources but does not adjust its internal clock. Consequently, the Primary servers converge to the Reference server's time. Having a single official time source allows you to maintain an accurate time using the server's internal hardware clock or such devices as radio clocks or modems. A network can have only one Reference server.

Single Reference Server

A Single Reference server is a Reference server that can set its synchronization flag to *true* even though there are no other servers with which it can synchronize. This type of time source is useful for networks that need only one time source.