Chapter 6

Managing NDS Entries

Overview

This chapter discusses how NDS manages entries in the Directory. It details the following operations and how they are performed in the Directory:

- Adding an entry
- Checking login restrictions
- Closing an iteration
- Comparing values
- Creating an entry directory
- Reading an entry's attribute information
- Reading an entry's non-attribute information
- Removing an entry
- Searching for entries

You should read this chapter if you are not familiar with the basic entry management operations in NDS.

Concepts to Know

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

- NDS Schema
- Entry attributes, mandatory and optional
- NDS entries and object classes
- Container entries
- Leaf entries
- Directory tree structure
- Partitioning and replication

Managing Entries in the NDS Database

The NDS database is represented logically as a tree-like hierarchical structure containing various objects, or *entries*. Each entry in the Directory Information Tree (DIT), or logical tree, corresponds to physical entries, or instances of the object in the Directory Information Base (DIB), which is the physical representation of the directory. These entries may be replicated on several servers in the network.

NDS entries and the rules governing their functions and attributes are based on NetWare 3.x bindery objects and on the X.500 standards for directory services (for more information about the X.500 directory services standards, see Appendix A, "X.500 and NDS: How Close Are They?").

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

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. The Entry ID is a 32-bit opaque value that serves as a handle. 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.

The methods used for obtaining an Entry ID are discussed in Chapter 5, "Traversing the NDS Tree". The operations described in this chapter assume that the requesting module has already obtained the Entry ID.

Note Unless specified otherwise, the term *client* means any client requesting services from a Directory Services agent independent of the platform the client is running on. It is used here for convenience only.

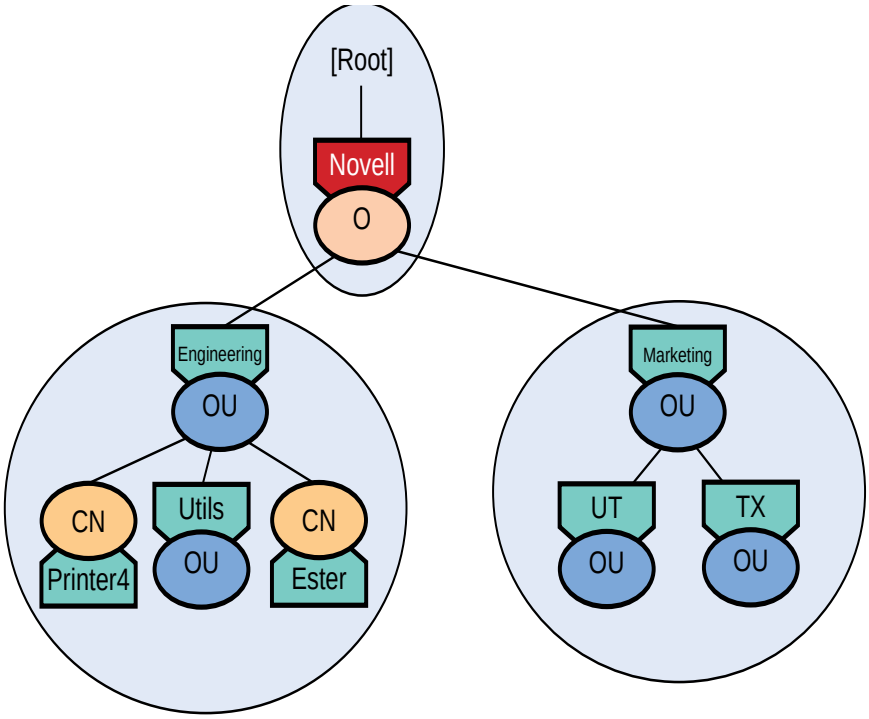
Adding an Entry

The *Add Entry* operation allows you to add an entry or an alias to the NDS Directory. 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 NDS 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

Figure 9.1 shows an entry for a User object, CN=Esther, that is being added as a child entry of OU=Engineering.

Figure 6.1
Adding an Entry to a Container



The mandatory attribute fields for the new entry must contain values, or the operation fails. In the example, the User entry CN=Esther 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, such as OU=Engineering in the example. 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. To add a new entry:

Table 6.1
Adding an Entry

Module	Action
Requesting module	1. Sends the server a *Add Entry* request.

Table 6.1
Adding an Entry (Continued)

Module	Action
Server	2. Checks that the object exists. 3. Checks that the object class is valid by: • Checking that the entry has only one value in the Object Class attribute. • Verifying that the value in the Object Class attribute is a valid object class. 4. If the entry is an alias entry, checks the *Aliased Name* attribute for the real entry's name by: • Checking that the alias has an *Aliased Object Name* attribute. • Checking that the alias does not refer to another alias. • Getting the aliased entry's base class. 5. Checks for attributes with special access control (read-only, hidden, and write-managed attributes). • Rejects the operation if the entry is hidden or read only, or if the client is not the server. 6. Checks that the replica is writable. 7. Checks that the client has the right to add the entry. 8. Adds the entry under the container entry. 9. Creates an entry directory if the entry is a Print Server, a Queue, or a subclass of Print Server or Queue. 10. Returns an *Add Entry* reply containing a completion code indicating success or failure.

Closing an Iteration

The *Close Iteration* operation is used to free memory used by an iteration handle. To close an iteration:

Table 6.2
Closing an Iteration

Module	Action
Requesting module	1. Sends a *Close Iteration* request to the server. This request contains an iteration handle.
Server	2. Frees the memory, unless DSV_LIST_PARTITIONS is the verb used. 3. Returns a *Close Iteration* reply containing a completion code that indicates success or failure.

Comparing 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 (for information about matching rules, see *NDS Schema Specification*). Therefore, the matching rules must provide for this operation.

To compare NDS attributes:

Table 6.3
Comparing Values

Module	Action
Requesting module	1. Sends a *Compare* request to the server.
Server	2. Verifies that it holds a readable replica. 3. Checks that the client has *Compare* rights on the attribute. 4. Compares the specified attribute to the attribute on the entry. 5. Returns a *Compare* reply indicating either *true* or *false.*

Getting a Count of Entries in a Partition

This operation allows a client to get a count of entries in a particular partition. To get a partition entry count:

Table 6.4
Getting a Partition Entry Count

Module	Action
Requesting module	1. Sends a *Get Partition Entry Count* request to the server.
Server	2. Locates all entries. 3. Returns a *Get Partition Entry Count* reply containing a count of the entries in the partition.

Getting the Replica Root Entry's ID

This operation lets a client find the Entry ID of the root entry in a particular replica of a partition. To obtain the replica root entry's Entry ID:

Table 6.5
Getting the Replica Root ID

Module	Action
Requesting module	1. Sends a *Get Replica Root ID* request to the server.
Server	2. Lcoates the entry record. 3. Finds the partition's root entry. 4. Returns a *Get Replica Root ID* reply containing a completion code and the Entry ID of the partition root.

Getting a Server's Network Address

This operation allows a requesting module to obtain a server's network address. Among other uses, the authentication process uses it to provide the client with the server's name. To get a server's network address:

Table 6.6
Getting a Server's Network Address

Module	Action
Requesting module	1. Sends a *Get Server Net Address* request to the server.
Server	2. Returns a *Get Server Net Address* reply containing the server's network address.

Listing Containable Classes

In the NDS schema, entries can only be contained in a container indicated by the containment rules for their base object class. *List Containable Classes* allows a client to specify a certain container and find out which object classes it can contain. For example, using *List Containable Classes*, a client could specify an Organizational Unit, Sales.Provo.Novell, and find out which object classes have Organizational Unit as their containment.

To list containable classes:

Table 6.7
Listing Containable Classes

Module	Action
Requesting module	1. Sends a *List Containable Classes* request to the server.
Server	2. Checks that the client has Browse [Entry] rights to the specified entry. 3. Goes through all schema classes and finds the those whose containment includes the base class of the entry specified. 4. Returns a *List Containable Classes* reply.

Modifying Entries

Modify Entry 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.

Values can be replaced using a combination of *DS_ADD_VALUE* and *DS_REMOVE_VALUE* changes. If any one of the modifications fails, the operation fails. The entry is left in the state it was before the attempted operation.

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.

To modify an entry:

Table 6.8
Modifying an Entry

Module	Action
Requesting module	1. Sends a *Modify Entry* request to the server.
Server	**Note** Steps 2 through 6 are repeated for each attribute (AVA).s 2. Checks that the client has *Read, Write, or Self* rights to the entry for each modification requested. 3. Checks attributes that are defined as *SF_WRITE_MANAGED.* • If removing such an attribute, the client checks for *Write* rights on the attribute. • Verifies that the client manages the entry being added or deleted as an attribute value. 4. Checks that the client has *Write* rights; if not, checks that the client has *Self* rights and if the attribute value refers to the client’s own entry. 5. If the client has no rights to read or write, returns *ERR_NO_SUCH_VALUE;* if the client has Read rights but not Write rights, returns *ERR_NO_ACCESS.* 6. Checks for the following attributes: • Security Equals. • Login Disabled. • Equivalent to Me. 7. Checks that the replica is writable. 8. Does one of the following: • If the modification is allowed, it modifies the entry and sends a *Modify Entry* reply indicating success. • If the modification fails or is not allowed, it returns a *Modify Entry* reply indicating failure. 9. If the operation was successful, does one or more of the following: • If a *Security Equal* attribute was modified, updates the security equivalences for each entry being modified. • If login is being disabled (the *Login Disabled* attribute is changed to 1), disables the logged-in user’s account. • If the *Equivalent To Me* attribute was modified, reports *DSE_UPDATE_SEV* event.

Modifying Relative Distinguished Names

To change the Relative Distinguished Name (RDN) of an entry, the client uses *Modify 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. 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 by sending 0 in the *deloldrdn* (Delete Old RDN) field. 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.

To modify a Relative Distinguished Name:

Table 6.9
Modifying a Relative Distinguished Name

Module	Action
Requesting module	1. Sends a *Modify RDN* request to the server.
Server	2. Checks that it holds a writable replica. 3. Verifies that the client has *Rename* rights to the entry. 4. Returns a *Modify RDN* indicating success or failure.

Moving an Entry

This 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.

For example, in the illustration below, the entry CN=Esther is being moved from the Engineering partition to the Marketing partition. Originally, the entry's Distinguished Name is Esther.Engineering.Novell. After the move, the Distinguished Name will be Esther.Marketing.Novell, reflecting the entry's new location in the Directory tree.

Figure 6.2
Moving a Leaf Entry

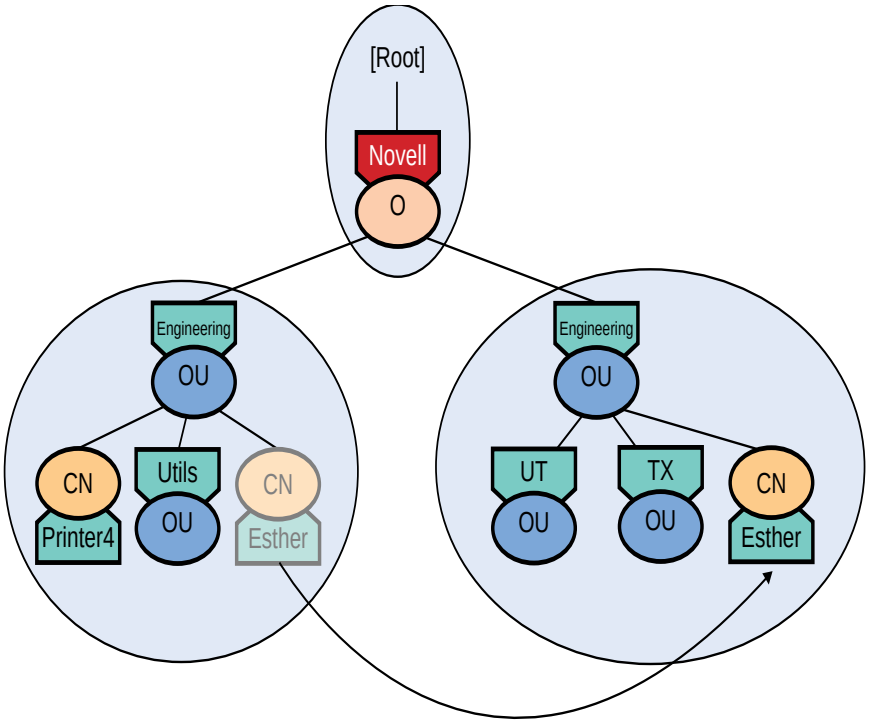
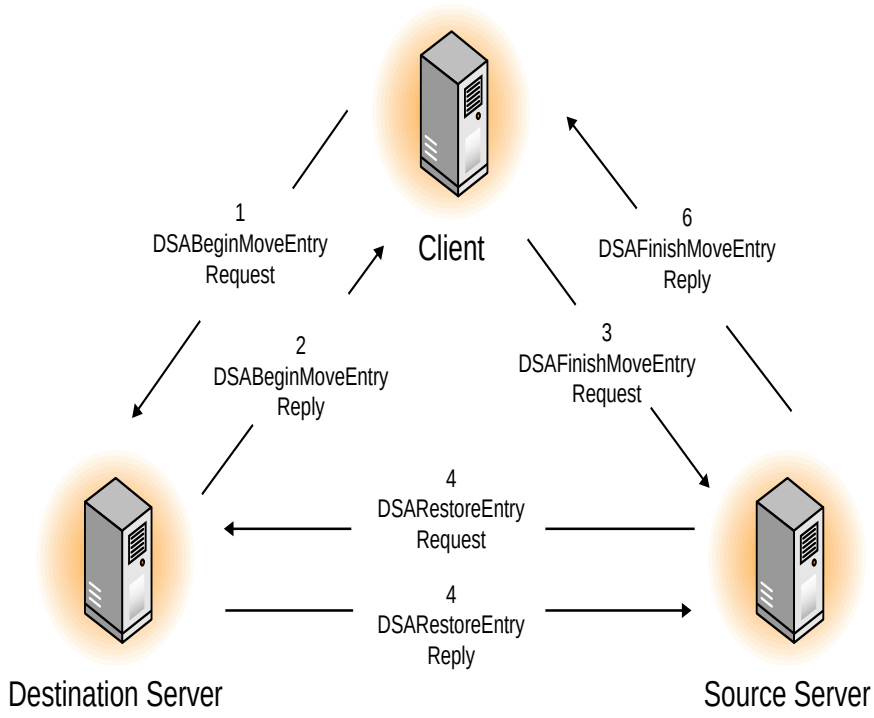


Figure 6.3 illustrates the three network entities involved in moving an entry: the client requesting the operation; the source server that holds the master replica of the entry before it moves; and the destination server that holds the master replica of the container under which the entry is moving.

Figure 6.3
Server Interactions in Moving an Entry



The entry to be moved must be a leaf entry or a container entry with no entries in it. To move an entry:

Table 6.10
Moving a Leaf Entry

Module	Action
Client	1. Sends a *Begin Move Entry* request to the destination server.
Destination Server	2. Verifies that its replica of the partition is not busy. 3. Checks to see that the client has *Add* rights to the destination parent container. 4. Verifies that it holds the master replica of the destination partition. 5. Checks that an entry does not exist with the moving entry's name. 6. Stores the following information in an *EXPECTATION* list in memory. 7. Returns a *Begin Move Entry* reply indicating success or failure (0=success).
Client	8. Sends the source server a *Finish Move Entry* request.

Table 6.10
Moving a Leaf Entry (Continued)

Module	Action
Source Server	9. Verifies that the source partition is not busy. 10. Checks to see that it holds the master replica of the source partition. 11. Verifies that the client has *Delete* rights to the source entry on the source server. 12. Verifies that the entry does not have an OBT_INHIBIT_MOVE obituary. 13. Checks that the entry's child entries are not involved in a *Move* operation. 14. Checks the entry's subordinate count and entry flags to see if it is a partition root or has subordinate entries. • If the entry is a partition root or has no subordinate entries, the source server calls the *Move SubTree* routine (see Chapter 6, "Managing Partitions"). • If the entry is not a partition root, the algorithm proceeds to the next step. 15. Locks the source partition. 16. Sends the destination server a *Restore Entry* request. *Restore Entry* places the entry in its new location on the destination server.
Destination server	17. Creates the entry for the destination entry (either an external reference or a real entry).
Source server	18. Swaps the entry IDs, so the the destination entry has the old ID and the source entry has the new ID, and so all values in the name base point to the current entry. 19. Adds an *OBT_MOVED* obituary to the source entry. 20. Clears the creation and modification times. 21. Subtracts 1 from the source parent's subordinate count. 22. Schedules the back link process. This updates all external references. 23. Removes the source entry's attribute values. 24. Adds the source entry to its in-memory list of entries that are moving. 25. Returns a *Finish Move Entry* reply indicating success or failure (0=success).
Destination server	26. Returns a *Restore Entry* reply.
Synchronization process	27. Propagates obituary to all replicas.

Opening a Stream Attribute

Stream attributes, such as login scripts, are stored in files separate from the NDS files database files. Normal entry access operations do not access these attributes; instead, remote file access operations access them. The *Read Entry* operation allows a client to discover the stream attributes,

and the *Open Stream* allows the client to open the stream. To open a stream attribute:

Table 6.11
Opening a Stream Attribute

Module	Action
Requesting module	1. Sends an *Open Stream* request to the server.
Server	2. Checks that it has a readable copy of the entry. 3. Checks that the client has Read or Write rights to the attribute, depending on the stream operation. • If the file doesn't exist, the server creates the file and then opens it using the requested access (read, write, or both). • If opening the file to write, the server registers with the OS for a close file event. 4. Obtains the file size. 5. Returns a *Open Stream* reply containing a file handle (the *Self* field from the value record) and the file size. 6. When the stream is closed, updates the entry's modification time stamp, which triggers synchronization.

Reading an Entry's Attribute Information

The *DSARead* operation returns information about an entry stored in the Directory. It reads attribute information associated with the entry. The attributes the entry has depend on the object class.

The client can choose to return information about:
- Specific attributes or a complete list of an entry's attributes.
- Attribute names only or attributes names and values.
- Effective access control privileges of the current user or of another user.

The granularity of the *Read* reply is the attribute value. An attribute value cannot be split across NDS iterations, but an attribute's list of values can be split. When split, each NDS reply message contains the *Syntax ID*, *Attribute Name*, and *Number of Values* fields. The *Number of Values* field indicates how many values are in the current reply.

The Basic *Read* Operation

To read an entry's attributes and values:

Table 6.12
Reading an Entry's Attributes and Values

Module	Action
Requesting module	1. Sends a *Read* request to the server.

Table 6.12
Reading an Entry's Attributes and Values

Module	Action
Server	2. Checks that the server holds a readable copy of the entry. 3. Fills the request. 4. Returns a *Read* reply to the client.

Attribute. In the *Read* reply, this field contains the information about each attribute returned in the NDS reply. It has the following wire format:

Table 6.13
Attribute Wire Format

Field	Type	Description
Align4		
Syntax ID	uint32	
Attribute Name	Ustring	The name of the attribute.
Align4		This field and the following two are present only if values are being sent.
Number of Values	uint32	The number of values in the following field.
Attribute Values	Attribute Result	See "Attribute Result" below.

Attribute Result. This field contains the attribute values returned in the NDS reply. It has the following structure:

Table 6.14
Attribute Result

Field	Type	Description
Align4		
Attribute Value	Determined by Syntax ID.	Contains an NDS attribute value.

Reading an Entry's Attribute Names and Values

Sometimes, a request is made for the attributes and values of a specific entry. If the Information Type field contains 01h (DS_ATTRIBUTE_VALUES), the client response contains the following attribute information, in addition to the other reply fields:

Table 6.15
***Attribute* Field (Attribute Names and Values)**

Field	Type	Description
Align4		
Syntax ID	uint32	
Attribute Name	Ustring	The name of the attribute returned.
Number of Values	uint32	The number of values in the following field.
Attribute Values	Attribute Result	Contains an Align4 and the attribute value.

The *Attribute Result* field contains the information about each attribute returned in the NDS reply. In this operation, it contains an align4 and the attribute value.

Reading an Entry's Attribute Names

If the Information Type field contains 00h (DS_ATTRIBUTE_NAMES), this *DSARead* operation returns only the names of NDS attributes an entry has.

In the reply, the resulting *Attribute* field has the following wire format for this operation:

Table 6.16
Attribute Format (Attribute Names Only)

Field	Type	Description
Align4		
Attribute Name	Ustring	The attribute name.

Reading a Trustee's Effective Rights

This operation lists the effective rights a trustee (indicated by the Distinguished Name) has to access the specified attributes on the entry. If the Information Type field contains 02h. DS_EFFECTIVE_ PRIVILEGES, the response contains the following attribute information:

Table 6.17
Attribute (Read Effective Rights)

Field	Type	Description
Align4		
Syntax ID	uint32	8 (SYN_INTEGER).
Attribute Name	Ustring	The name of the attribute whose effective rights are returned.
Number of Values	uint32	1.
Length of Attribute Value		4.

Table 6.17
Attribute (Read Effective Rights) (Continued)

Field	Type	Description
Privileges	uint32	The subject's privileges. These are encoded as described in Chapter 4, "NDS Services and Supporting Services".

Reading an Entry's Values and Value Information

NDS allows clients and servers to read information about attribute values in addition to reading the values. If the Information Type field contains 03h (DS_VALUE_INFO), the following attribute information is retuned:

Table 6.18
Attribute Format (Read Values and Value Information)

Field	Type	Description
Align4		
Syntax ID	Varies	Depends on the attribute's syntax.
Align4		
Attribute Name	Ustring	The name of the attribute.
Align4		
Number of Values	uint32	The number of values returned.
Attribute Information	Attribute Result	See "Attribute Result" below.

In the reply, the *Attribute Result* field contains the following information:

Table 6.19
Attribute Result (Read Values and Value Information)

Field	Type	Description
Value Flags	uint32	00000001h. DS_NAMING (The entry's naming value). 00000002h. DS_BASECLASS (The value is the entry's base class). 00000004h. DS_PRESENT (Value is present).
Modification Time	Time Stamp	The attribute's creation time.
Attribute Value	Depends on syntax	Depends on the syntax ID.

Reading an Entry's Abbreviated Values and Value Information

NDS allows clients and servers to read additional information about attribute values. This operation does not return the values but reports their lengths and other information. If the Information Type field contains 04h

(DS_ABBREVIATED_VALUE), the reply contains the following attribute information:

Table 6.20
Attribute Format (Read Abbreviated Values and Value Information)

Field	Type	Description
Align4		
Syntax ID	Depends on attribute	
Attribute Name	Ustring	The name of the attribute returned.
Align4		
Number of Values	uint32	The number of values returned.
Attribute Information	Attribute Result	See below.

In the *Read* reply for this operation, the *Attribute Result* field contains the following information:

Table 6.21
Attribute Result (Read Abbreviated Values and Value Information)

Field	Type	Description
Value Flags	uint32	00000001h. DS_NAMING (The entry's naming value). 00000002h. DS_BASECLASS (The value is the entry's base class). 00000004h. DS_PRESENT (Value is present).
Modification Time	Time Stamp	The last time the value was modified.
Attribute Length	uint32	Length of the value.

Reading an Entry's Non-Attribute Information

Servers and clients use the *Read Object Info* operation to obtain information about the entry other than its attribute values. Clients can use information flags to filter the information returned in several ways.

To read an entry's non-attribute information:

Table 6.22
Reading an Entry's Non-Attribute Information

Module	Action
Requesting module	1. Sends a *Read Object Info* request to the server.
Server	2. Fills the request. 3. Returns a *Read Object Info* reply containing the information specified in the Information Flags.

Removing an Entry

Remove Entry removes an entry from the Directory tree. The entry being removed must be a leaf or a container 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. If the entry ID indicates an alias entry, the operation removes the alias, not the entry the alias refers to.

To remove an entry:

Table 6.23
Removing an Entry

Module	Action
Requesting module	1. Sends a *Remove Entry* request to the server. 2. Verifies that the replica is writable.
Server	3. Checks that the client has *Delete* rights to the entry. 4. Checks the entry flags to verify that the entry is not a partition root. 5. Checks to see if the entry is a print queue or print server. 6. Removes the related directory if the entry is a queue or print server. If the entry is a queue, the operation uses Host Server and Queue Directory; if the entry is a print server, it uses Host Device. 7. Removes the entry and revokes file system rights (removes it from the file system trustee node).

Saving a Database Set

This operation allows a client to save a database set (DIB set) with a file extension. A database set is a copy of the NDS database files. To save a database set:

Table 6.24
Saving a Database

Module	Action
Requesting module	1. Sends a *Save DIB Set* request to the server.
Server	2. Saves the data set in SYS:_NETWARE with the file extension requested. 3. Returns a *Save DIB Set* reply indicating success or failure.