NDS and Bindery Service Group

NDS/Bindery Overview

NDS

NDS™ is a complex database system that allows information to be stored, replicated, and accessed across an entire network or intranetwork. NDS is available on NetWare® versions 4.0 and higher. To access infomation about NDS development, begin with NDS: Guides.

Bindery

The bindery is a flat, server-centric database for managing objects on NetWare 2.x and 3.x servers. To access information about developent for the bindery, begin with Bindery: Guides.

Bindery

Bindery: Guides

NetWare® 3.x servers use the bindery database to identify and store information about network objects. Bindery functions enable applications to read and modify standard information stored in the bindery and to create and manage their own Bindery object data.

Bindery Overview

Bindery: Task Guide

Bindery: Concept Guide

Bindery: Functions

Bindery: Task Guide

Bindery Objects

Creating a Bindery Object

Scanning for Bindery Objects

Scanning the Bindery: Example

Bindery Properties

Checking for a Member of a Set Property

Reading the Value of a Bindery Property

Scanning Bindery Properties

Scanning the Bindery: Example

Bindery: Concept Guide

General Information

Bindery Overview

Bindery vs. NDS

Bindery Files

Bindery Objects

Bindery Properties

Bindery: Functions

Bindery Objects

Bindery Objects

Object ID Bindery Parameter

Object Type Bindery Parameter

Object Name Bindery Parameter

Object Flags Bindery Parameter

Object Security Bindery Parameter

Bindery Object Security Table

Bindery Properties

Bindery Properties

Bindery Property Parameters

Property Name Bindery Parameter

Property Flags Bindery Parameter

Static and Dynamic Properties in a Bindery

Item Property in the Bindery

Set Property Bindery Parameter

Property Security Bindery Parameter

Bindery Properties Associated with NetWare Security

USER_DEFAULTS and LOGIN_CONTROL Bindery Properties

OLD_PASSWORDS Bindery Property

NODE_CONTROL Bindery Property

ACCT_LOCKOUT Bindery Property

Bindery Functions

Types of Bindery Functions

Bindery Status Functions

Bindery Object Functions

Bindery Property Functions

Bindery Password Functions

Bindery Object Information Functions

Wildcard Characters in Bindery Functions

Bindery: Tasks

Checking for a Member of a Set Property

The **NWIsObjectInSet** function provides a simple method for determining whether an object is a member of a set property without your having to read the property value. Specify the object name, type, and property to be searched and the name and type of the member object. The object is a member if the **NWIsObjectInSet** function returns successfully.

Related Topics:

Reading the Value of a Bindery Property

Creating a Bindery Object

When creating a bindery object, the client application is responsible not only for creating the object itself, but also for adding any associated properties and defining their values. The bindery does not check the state of the object or verify that it has the necessary property values.

To create an object:

1. **Call the NWCreateObject function.**

2. **Call the NWCreateProperty function.**

3. **Call the NWWritePropertyValue function.**

The **NWCreateObject** function assigns the object its name, type, object flags, and object security. The **NWCreateProperty** function adds associated properties to the object. As with the object definition, you must specify the name, flags, and security for each object. The **NWWritePropertyValue** function assigns specific values to the object.

Related Topics:

Scanning for Bindery Objects

Reading the Value of a Bindery Property

To read a property, call the **NWReadPropertyValue** function (no wild characters are allowed). Set properties are assumed to be arrays of bindery object IDs and are returned accordingly. No assumptions are made about item properties. If a property value exceeds 128 bytes, you must call the **NWReadPropertyValue** function iteratively and read the value in 128-byte segments.

Related Topic:

Checking for a Member of a Set Property

Scanning Bindery Properties

Scanning Bindery Properties

To find the properties of an object or to verify that a particular property is assigned to an object, call the **NWScanProperty** function which allows you to match property names with wild characters.

To get information about all properties assigned to an object, call the **NWScanProperty** function iteratively:

1. **On the initial call, set the oject name and type appropriately, and set the search property name to the asterisk wild card.**
2. **On the initial call, also set the *iterHandle* parameter to -1.**
3. **When the function returns, check the *moreFlag* parameter.**

   If the *moreFlag* parameter is set to 0xFF, there are more properties. The *iterHandle* parameter will be set in preparation for the next call to the **NWScanProperty** function.

   If there are no more properties, the *moreFlag* parameter is set to 0.

Related Topics:

Checking for a Member of a Set Property

Reading the Value of a Bindery Property

Scanning the Bindery: Example

Scanning for Bindery Objects

The **NWScanObject** function lets you scan the bindery of a server for objects matching a specified name and type. For each match, the **NWScanObject** function returns the has-properties flag, object flags, and object security field for the object. It also returns the name and type of the next matching object.

The name can be expressed using the asterisk (*) and question mark (?) as wild characters. The asterisk matches 0 or more characters. The question mark matches exactly one character. Below are some matching examples:

"*"	Matches any object name.
"S*"	Matches any object name beginning with S.
"??"	Matches any two-character object name.

The type can be any object type including the wild type, 0xFFFF.

To scan the bindery for multiple objects, call the **NWScanObject** function iteratively until it returns NO_SUCH_OBJECT. This error indicates no more objects matching the specified name and type exist in the bindery.

Related Topics:

Creating a Bindery Object

Scanning the Bindery: Example

Setting Bindery Emulation

For bindery emulation to be operative, two conditions must be met:

Bindery emulation must be set

The server must have a valid replica of the organization to be emulated in bindery (flat) mode

To set bindery emulation, include the following line in the autoexec.ncf file:

`set Bindery Context = o = MyOrg` (where "MyOrg" is a organization having a valid replica on the server being set).

Bindery context can also be set from the server command line with the same command, but including the command in the autoexec.ncf file is preferred.

> **NOTE:** It is possible to enter the command with an invalid organization. The command returns a message that bindery context is set to the invalid organization, but only the bindery string is set---bindery emulation remains unset. Any calls to bindery functions then fail and return errors.
>
> To verify that bindery emulation is actually set, at the server command prompt, type `config`. The displayed configuration includes a report of all valid bindery emulation settings.

For further information on bindery emulation setting, consult the NetWare® server documentation.

Bindery: Concepts

ACCT_LOCKOUT Bindery Property

ACCT_LOCKOUT specifies intruder lockout values. Intruder detection is active only if this property is assigned to the server object and is in the following format:

	Size	**Field: Description**
	nuint16	allowed login attempts: Maximum number of incorrect login attempts before intruder detection is in effect. 0 causes intruder detection on the first bad login attempt.
	nuint16	reset minutes: Number of minutes that must pass without a bad login attempt before the bad login attempts field in the LOGIN_CONTROL property is reset to 0
	nuint16	lockout minutes: Number of minutes an account should remain locked if an intruder is detected. Nonzero indicates the account is locked.

Related Topics:

Bindery Properties Associated with NetWare Security

Activity Coordination

Bindery makes no attempt to coordinate activities among multiple stations that concurrently read or write data to a single property. One workstation might read a partially updated property and get inconsistent data if the data of the property extends across multiple segments. Coordination on Reads and Writes must be handled by application programs. Logical record locks can be used to coordinate activities among applications.

Bindery Files

Each NetWare server maintains a bindery database. One server cannot coordinate its bindery data with other servers on the network. For NetWare 2.2, the bindery database consists of two files:

net$bind.sys

net$bval.sys

For NetWare 3.11, the bindery consists of three files:

net$obj.sys

net$prop.sys

net$val.sys

Bindery files are hidden in the SYS:SYSTEM directory.

Bindery allows applications to open and close the bindery so that these files can be archived, but an application must have supervisor equivalence to open and close the bindery. When the files are closed, many server operations become disabled so take precautions. NetWare 4.x Bindery will not open or close the bindery.

Related Topics:

Bindery: Concept Guide

Bindery Object Functions

These functions perform operations on bindery objects:

NWChangeObjectSecurity
NWCreateObject
NWDeleteObject
NWRenameObject
NWScanObject

Related Topics:

Types of Bindery Functions

Bindery Object Information Functions

These functions return information related to bindery objects:

NWGetBinderyAccessLevel
NWGetObjectDiskSpaceLeft
NWGetObjectEffectiveRights
NWGetObjectID
NWGetObjectName
NWScanObjectTrusteePaths

Related Topics:

Types of Bindery Functions

Bindery Object Properties

Each bindery object can be assigned one or more properties. Properties identify categories of information associated with an object. For example, a user object has a GROUPS_I'M_IN property, an ACCOUNT_BALANCE property, and a PASSWORD property. Each property provides storage space appropriate to the associated values. All properties are assigned the following information:

Property Name

Property Flags

Property Security

Related Topics:

Property Name Bindery Parameter

Property Flags Bindery Parameter

Property Security Bindery Parameter

Bindery Objects

Bindery Objects

All bindery objects must be assigned the following information:

Object ID

Object Type

Object Name

Object Flags

Object Security

Has-Properties Flag

Related Topics:

Extended Object Type Values

Object ID Bindery Parameter

Object Type Bindery Parameter

Object Name Bindery Parameter

Object Flags Bindery Parameter

Object Security Bindery Parameter

Has-Properties Flag Bindery Parameter

Bindery: Concept Guide

Bindery Overview

NetWare® 3.x servers use the bindery database to identify and store information about network objects. Bindery allows applications to read and modify standard information stored in the bindery and to create and manage their own object data.

The bindery serves many purposes. It is the basis for identifying users of the file system, both through login control and file trustee rights. Users, user groups, print servers, and other objects that require access to the NetWare file system must be represented in the bindery.

Another use of the bindery is network advertising. The bindery advertises services and circulates their network addresses to NetWare servers and other network services.

A third role for the bindery is storing application-specific data. For example, applications often use the bindery to maintain lists of users that can access the application services.

Bindery Password Functions

These functions perform operations on object passwords:

NWChangeObjectPassword

NWDisallowObjectPassword
NWVerifyObjectPassword

Related Topics:

Types of Bindery Functions

Bindery Properties

The standard properties defined by NetWare for managing user and group access to the NetWare server are defined as follows:

	Name	**Flags**	**Object Type: Description**
	ACCOUNT_BALANCE	static/item	user
	ACCOUNT_HOLDS	dynamic/item	user
	ACCOUNT_SERVERS	static/set	server
	ACCT_LOCKOUT	static/item	server
	BLOCKS_READ	static/item	server
	BLOCKS_WRITTEN	static/item	server
	CONNECT_TIME	static/item	server
	DISK_STORAGE	static/item	server
	GROUP_MEMBERS	static/set	user group: List of users that are

			members of a user group
	GROUPS_I'M_IN	static/set	user: List of user groups of which a user is a member
	IDENTIFICATION	static/item	user: User of users group full name
	LOGIN_CONTROL	static/item	user
	NET_ADDRESS	dynamic/item	server
	NODE_CONTROL	static/item	user
	OLD_PASSWORDS	static/item	user
	OPERATORS	static/set	server: List of objects that are console operators
	PASSWORD	static/item	user: Encrypted password of an object
	Q_DIRECTORY	static/item	queue
	Q_OPERATORS	static/set	queue
	Q_SERVERS	static/set	queue
	Q_USERS	static/set	queue
	REQUESTS_MADE	static/item	server

	SECURITY_EQUALS	static/set	user: List of objects that are equivalent to the associated object
	USER_DEFAULTS	static/item	supervisor

Related Topics:

Bindery: Concept Guide

Bindery Properties Associated with NetWare Security

The following properties are associated with NetWare security:

LOGIN_CONTROL

OLD_PASSWORDS

NODE_CONTROL

ACCT_LOCKOUT

Related Topics:

ACCT_LOCKOUT Bindery Property

USER_DEFAULTS and LOGIN_CONTROL Bindery Properties

OLD_PASSWORDS Bindery Property

NODE_CONTROL Bindery Property

Bindery Properties

Bindery Property Functions

These functions operate on bindery properties:

NWAddObjectToSet
NWChangePropertySecurity
NWCreateProperty
NWDeleteObjectFromSet

NWDeleteProperty
NWIsObjectInSet
NWReadPropertyValue
NWScanProperty
NWWritePropertyValue

Related Topics:

Types of Bindery Functions

Bindery Status Functions

Bindery Status Functions open and close the bindery files on a specified server and are not supported by NetWare 4.x Bindery:

NWCloseBindery
NWOpenBindery

Related Topics:

Types of Bindery Functions

Bindery vs. NDS

The NetWare 4.0 OS replaced the bindery with NDS (a new object database). NDS offers many advantages over the bindery, including a hierarchical structure and global naming.

However, to maintain compatibility with bindery-based servers and to work effectively with such NetWare features as file trustee rights, NDS provides built-in bindery context. This is provided by a bindery-like database maintained by NDS for objects contained in the local directory partitions of a server.

NDS generates object IDs through NetWare 4.x Bindery and makes them available to Bindery clients using the local file system, queue management system, and other bindery-oriented services. These values, however, are dynamic, not remaining consistent over time. NDS object IDs and the object IDs returned by Bindery are the same.

Related Topics:

Bindery: Concept Guide

Extended Object Type Values

Use the Wild object type (0xFFFF) when scanning the bindery.

	Pascal	**Values**
0x2	$2	OT_TIME_SYNCHRONIZATION_SERVER
0x2E00	$2E00	OT_ARCHIVE_SERVER_DYNAMIC_SAP
0x4700	4700	OT_ADVERTISING_PRINT_SERVER
0x5000	$5000	OT_BTRIEVE_VAP
0x5300	$5300	OT_PRINT_QUEUE_USER

Has-Properties Flag Bindery Parameter

The *hasPropertiesFlag* parameter is a single-byte flag indicating whether any properties are associated with the object:

00h	No properties
FFh	1 or more properties

Related Topics:

Bindery Objects

Item Property in the Bindery

The *item property* parameter is a 128-byte string and can take up as much of this space as necessary. For example, the property ACCOUNT_BALANCE is an item property that contains a monetary balance in the first few bytes. The remainder of the 128 bytes is zeroed out and must be interpreted by the application.

Related Topics:

Property Flags Bindery Parameter

USER_DEFAULTS and LOGIN_CONTROL Bindery Properties

The USER_DEFAULTS property is used by system utilities to initialize the LOGIN_CONTROL property.

The LOGIN_CONTROL property tracks the current state of security for the associated account and contains the following fields:

	Size	Field: Description
	nuint8[3]	account expiration date: Date the account expires
	nuint8	account disabled flag: Whether the accound is enabled (0x00) or disabled (0xFF). NetWare checks the value every half hour. If set to 0xFF since a user logged into an account, NetWare asks the user to log out. Within five minutes, NetWare will clear the connection.
	nuint8[3]	password expiration date: Date the password expires

	nuint8	grace logins remaining: Number of times a user can log in using an expired password. If set to 0xFF, it is not decremented. Otherwise, it is decremented each time the user logs in after the password expired. If set to 0, the user cannot log in.
	nuint16	password expiration interval: Number of days between password expiration dates. In 4.x, it must be nonzero to check the old password list when changing a password.
	nuint8	grace login reset value: Reset value for the grace logins remaining field after a password change
	nuint8	mininum password length: Minimum length permitted for a password. If 0, no password is needed. In 4.x, it must be nonzero to check the old password list when changing a password.
	nuint16	maximum concurrent connections: Maximum conconcurrent connections allowed per user. If 0, the limit is the maximum supported by the server.
	nuint8[42]	allowed login time bit map: 336 half-hour periods during a week when a user can login to the server. Bit 1 of byte 0 represents 12:00 to 12:29 am Sunday. Bit 2 of byte 0 is 12:30 to 12:59 am Sunday. Setting the bit permits logins during the corresponding

		period.
	nuint[6]	last login date and time: Most recent time the user logged in
	nuint8	restriction flags: Who is allowed to change the password on the account: 0x00 Anyone, 0x01 Supervisor. 0x02 is set if the OLD_PASSWORDS property exists for the account. In 3.x and 4.x, it must be 0x02 to keep an old passwords list.
	nuint8	reserved
	nuint32	maximum disk usage (in blocks): Number of 4K disk blocks that may be used by the account. If 0x7FFFFFFF, there is no limit.
	nuint16	bad login count: Number of bad login attempts since the last reset time. (Intruder detection is active only if the ACCT_LOCKOUT property is assigned to the file server object). The field is reset when a successful login occurs or the number of minutes in the *reset minutes* parameter of ACCT_LOCKOUT expires. If the account is locked because of intruder detection, it is set to 0xFFFF.
	nuint32	next reset time: Time when the bad login count should be set to 0

	n ui nt 8[12]	bad login address: Address of the station making the last bad login attempt or provoked an account lockout

For all 3-byte date values, the first byte contains the year (0=1900, 1=1901, 2=1902, etc.), the second byte contains the month, and the third byte contains the day. No date is defined if all three bytes are 0.

Related Topics:

Bindery Properties Associated with NetWare Security

Maximum Rights Mask Values

TA_OPEN is obsolete in 3.x and above.

	Bit
	TA_NONE
	TA_READ
	TA_WRITE
	TA_OPEN
	TA_CREATE
	TA_DELETE
	TA_OWNERSHIP
	TA_SEARCH
	TA_MODIFY
	TA_ALL

NODE_CONTROL Bindery Property

The NODE_CONTROL property is a list of network addresses from which the account can login. The addresses are 10 bytes each (a 4-byte network address and 6-byte node address). Each data segment holds up to 12 addresses. (The last 8 bytes of each segment are not used.) The list terminates with a zero address or with the last data segment. If a node address is set to 0xFFFFFFFFFF, the account can log in from any station.

Related Topics:

Bindery Properties Associated with NetWare Security

Object Flags Bindery Parameter

The *objectFlags* parameter is a single-byte flag specifying whether the object is static:

00h	Static
01h	Dynamic

A static object exists in a bindery until an application intentionally deletes it. A dynamic object disappears from a network server bindery when the network server is rebooted.

Objects placed in the bindery by SAP are dynamic. The NetWare server closely monitors such objects. The service provider must advertise periodically or the NetWare server deletes the object.

Related Topics:

Bindery Objects

Object ID Bindery Parameter

The *objectID* parameter is a 4-byte number uniquely identifying the object within a particular network server bindery and is not recognized by other NetWare servers. The NetWare operating system (not the application) will assign this number.

Related Topics:

Bindery Objects

Object Name Bindery Parameter

The *objectName* parameter is a 48-byte string (including a NULL terminator) containing the name of the object. The name consists of only printable characters and cannot include spaces or any of the following characters:

/	slash
\	backslash
:	colon
;	semicolon
,	comma
*	asterisk
?	question mark

IMPORTANT: Object names are recorded in uppercase in the bindery.

Related Topics:

Bindery Objects

Object Security Bindery Parameter

The *object security* parameter is a single-byte flag that determines object access. The low-order nibble determines who can read (scan for and find) the object. The high-order nibble determines who can write to the object (add or delete properties). The table below shows the values defined for each nibble.

Table auto. Bindery Object Security Table

Bit	Security Level	Description
0x 00	BS_ANY_READ	Anyone can read the object, even users who aren't logged in.
0x 01	BS_LOGGED_REA D	Only clients logged in to the server can read the object.
0x 02	BS_OBJECT_READ	Only clients logged in to the server with this object's name, type, and password can read the object.
0x 03	BS_SUPER_READ	Only clients with supervisor equivalence can read the object.
0x 04	BS_BINDERY_REA D	Only the NetWare operating system can read the object.
0x 00	BS_ANY_WRITE	Anyone can modify the object, even users who aren't logged in.
0x 10	BS_LOGGED_WRI TE	Only clients logged in to the server can modify the object.
0x 20	BS_OBJECT_WRIT E	Only clients logged in to the server with this object's name, type, and password can modify the object.
0x 30	BS_SUPER_WRITE	Only clients with supervisor equivalence can modify the object.
0x 40	BS_BINDERY_WRI TE	Only the NetWare operating system can modify the object.

Parent Topic:

Bindery Objects

Object Type Bindery Parameter

The *objectType* parameter is a 2-byte number classifying the object. The bindery expects these values to be expressed in high-low format. The values shown are in high-low format and do not need to be swapped:

	Pascal	**Name**
0xFFFF	$FFFF	OT_WILD
0x0000	$0000	OT_UNKNOWN
0x0100	$0100	OT_USER
0x0200	$0200	OT_USER_GROUP
0x0300	$0300	OT_PRINT_QUEUE
0x040	$0400	OT_FILE_SERVER

0		
0x0500	$0500	OT_JOB_SERVER
0x0600	$0600	OT_GATEWAY
0x0700	$0700	OT_PRINT_SERVER
0x0800	$0800	OT_ARCHIVE_QUEUE
0x0900	$0900	OT_ARCHIVE_SERVER
0x0	$0	OT_JOB_QUEUE
0x0B00	$0B00	OT_ADMINISTRATION
0x2100	$2100	OT_NAS_SNA_GATEWAY
0x	$2	OT_REMOTE_BRIDGE_SERVER

2600	600	
0x27000	$2700	OT_TCPIP_GATEWAY

Related Topics:

Extended Object Type Values

OLD_PASSWORDS Bindery Property

The OLD_PASSWORDS property is a list of 8 previous passwords. The list is stored in the first data segment.

Related Topics:

Bindery Properties Associated with NetWare Security

Property Name Bindery Parameter

The *propertyName* parameter is a 15-byte string (including a NULL terminator) containing the name of the property. See Object Name Bindery Parameter.

IMPORTANT: Property names are recorded in uppercase in the bindery.

Related Topics:

Bindery Properties

Property Flags Bindery Parameter

The *propertyFlags* parameter is a single-byte flag with bits 0 and 1 defined. Bit 0 is the static/dynamic toggle flag, and bit 1 is the item/set toggle flag. The flags are combined so that both set and item properties can be static or dynamic. The bits are defined as follows:

Bit 0:

0 Static

1 Dynamic

Bit 1:
0 Item
1 Set

Related Topics:

Bindery Properties

Property Security Bindery Parameter

The *property security* parameter is a single-byte string that determines who can access the property. The low-order nibble determines who can scan for and find the property (read security). The high-order nibble determines who can add values to the property (write security). Possible values for this flag correspond to those defined for the object security field. See theObject Security Bindery Parameter table.

Related Topics:

Bindery Properties

Security Rights Mask Values

Read Value	**Write Value**	**Access: Description**
BS_ANY_READ	BS_ANY_WRITE	Anyone: Access allowed to all clients, even if the client has not logged in to the server
BS_LOGGED_READ	BS_LOGGED_WRITE	Logged: Access allowed to all clients logged in to the server
BS_OBJECT_READ	BS_OBJECT_WRITE	Object: Access allowed only to clients who have logged in to the server with the name,

		type, and password of the object
BS_SUPER_READ	BS_SUPER_WRITE	SUPERVISOR: Access allowed only to clients who have logged in to the server as SUPERVISOR, or as a bindery object that is security-equivalent to SUPERVISOR
BS_BINDERY_READ	BS_BINDERY_WRITE	NetWare: Access allowed only to NetWare

Values can be ORed together. A bindery object with BS_SUPER_WRITE ORed with BS_LOGGED_READ indicates any user logged in to the NetWare server can view an object or property, but only the SUPERVISOR can add or change a property.

The *newPropertySecurity*, *accessLevel*, *propertySecurity*, *objSecurity*, and *newObjSecurity* parameters are bytes in which the low nibble controls Read security and the high nibble controls Write security.

Set Property Bindery Parameter

The *setProperty* parameter contains a list of 1 to 32 object IDs in a 128-byte segment. Each object ID is 4 bytes. The property GROUPS_I'M_IN is an example of a Set property. This property contains the object IDs (from 1 to 32) of user groups to which the user belongs. The values of a Set property are always object IDs grouped into one or more 128-byte segments.

Related Topics:

Property Flags Bindery Parameter

Static and Dynamic Properties in a Bindery

The Static property is recorded in server memory and remains a part of the bindery until the property is explicitly deleted.

bindery until the property is explicitly deleted.

In contrast, the Dynamic property is created during the course of a bindery session and is deleted from the bindery of a network server when the server is rebooted.

Related Topics:

Property Flags Bindery Parameter

Types of Bindery Functions

The types of Bindery Functions follow:

Bindery Status Functions

Bindery Object Functions

Bindery Object Information Functions

Bindery Password Functions

Bindery Property Functions

Wildcard Characters in Bindery Functions

The asterisk matches 0 or more characters while the question mark matches exactly one character:

"*"	Matches any characters in a string (such as an object name)
"S*"	Matches any characters in a string beginning with S (such as a property name)
"??"	Matches any two characters

Bindery: Functions

NWAddObjectToSet

Adds a member to a bindery property of type SET
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWAddObjectToSet (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          propertyName,
   pnstr8          memberName,
   nuint16         memberType);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWAddObjectToSet
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   propertyName : pnstr8;
   memberName : pnstr8;
   memberType : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

objName

(IN) Points to the new SET object name.

objType

(IN) Specifies the SET object type.

propertyName

(IN) Points to the property name of the set.

memberName

(IN) Points to the name of the bindery object being added to the set.

memberType

(IN) Specifies the bindery object type of the member being added.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x89 96	SERVER_OUT_OF_MEMORY
0x89 E8	WRITE_PROPERTY_TO_GROUP
0x89 E9	MEMBER_ALREADY_EXISTS
0x89 EA	NO_SUCH_MEMBERS
0x89 EB	NOT_GROUP_PROPERTY
0x89 EC	NO_SUCH_SEGMENT
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F8	NO_PROPERTY_WRITE_PRIVILEGE
0x89 FB	NO_SUCH_PROPERTY
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

A client must have write access to the SET property to call **NWAddObjectToSet**.

The *objName*, *objType*, *propertyName*, *memberName*, and *memberType* parameters must uniquely identify the property and cannot contain wildcard characters.

NWAddObjectToSet searches consecutive segments of the property value for an open slot where it can record the unique bindery object identification of the new member and records the bindery object identification in the first available slot. If **NWAddObjectToSet** finds no available slot, a new segment is created, the new unique bindery object identification of the member is written into the first slot of the new segment, and the rest of the segment is filled with zeros.

NCP Calls

0x2222 23 65 Add Bindery Object To Set

See Also

NWDeleteObjectFromSet, **NWIsObjectInSet**

NWChangeObjectPassword

Changes the specified password of an object to a new password
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWChangeObjectPassword (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          oldPassword,
   pnstr8          newPassword);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWChangeObjectPassword
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   oldPassword : pnstr8;
   newPassword : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle corresponding to the server to receive the change.

objName

(IN) Points to the name of the object whose password is to be changed.

objType

(IN) Specifies the type of the object.

oldPassword

(IN) Points to the old password.

newPassword

(IN) Points to the new password.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x89 D7	PASSWORD_NOT_UNIQUE
0x89 D8	PASSWORD_TOO_SHORT
0x89 DC	ACCOUNT_DISABLED
0x89 DE	PASSWORD_HAS_EXPIRED_NO_GRACE
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 FB	INVALID_PARAMETER
0x89 FF	NO_SUCH_OBJECT_OR_BAD_PASSWORD

Remarks

NWChangeObjectPassword does not require the old password to be known.

For **NWChangeObjectPassword** to work properly, LOGIN_CONTROL must be set appropriately.

NWChangeObjectPassword attempts to change the password by using encryption. If the server does not support encryption, **NWChangeObjectPassword** attempts to change the password without using encryption.

Clients can change their own password. To change the password for other bindery objects, the client must be a SUPERVISOR or SUPERVISOR equivalent.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 23	Get Login Key
0x2222 23 53	Get Bindery Object ID
0x2222 23 75	Keyed Change Password

See Also

NWCreateObject, **NWCreateProperty**

NWChangeObjectSecurity

Changes the security access mask of a bindery object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWChangeObjectSecurity (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   nuint8          newObjSecurity);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWChangeObjectSecurity
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   newObjSecurity : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to a string containing the name of the object whose security is to be changed.

objType

(IN) Specifies the bindery object type.

newObjSecurity

(IN) Specifies the new security access mask for the specified object.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F1	INVALID_BINDERY_SECURITY
0x89 F5	NO_OBJECT_CREATE_PRIVILEGE
0x89 FC	NO_SUCH_OBJECT
0x98 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

The *objName* and *objType* parameters must uniquely identify the bindery object and cannot contain wildcard characters.

NWChangeObjectSecurity cannot set or clear bindery Read or Write security. Only SUPERVISOR or a bindery object that is security equivalent to SUPERVISOR can change security for a bindery object.

See Security Rights Mask Values.

See Object Type Bindery Parameter.

See Extended Object Type Values.

NCP Calls

0x2222 23 56 Change Bindery Object Security

See Also

NWGetObjectID

NWChangePropertySecurity

Changes the security access mask of a property in a bindery object on the NetWare server associated with the given connection identification

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWChangePropertySecurity (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          propertyName,
   nuint8          newPropertySecurity);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWChangePropertySecurity
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   propertyName : pnstr8;
   newPropertySecurity : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle on which the security property should be changed.

objName

(IN) Points to the name of the bindery object associated with the property whose security is being changed.

objType

(IN) Specifies the type of the object described by the *objName* parameter.

propertyName
(IN) Points to the name of the affected property.

newPropertySecurity
(IN) Specifies the new security access mask for the property.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F1	INVALID_BINDERY_SECURITY
0x89 F2	NO_OBJECT_READ_PRIVILEGE
0x89 F5	NO_OBJECT_CREATE_PRIVILEGE
0x89 F6	NO_PROPERTY_DELETE_PRIVILEGE
0x89 FC	NO_SUCH_PROPERTY
0x89 FC	NO_SUCH_OBJECT
0x98 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

NWChangePropertySecurity requires Write access to the bindery object and Read and Write access to the property.

The *objName*, *objType*, and *propertyName* parameters must uniquely identify the property and cannot contain wildcards.

NWChangePropertySecurity cannot set or clear bindery Read or Write security. The requesting process cannot change the security of a property to a level greater than the property access of the process.

See Security Rights Mask Values.

See Object Type Bindery Parameter.

See Extended Object Type Values.

NCP Calls

0x2222 23 59   Change Property Security

See Also

NWChangeObjectSecurity

NWCloseBindery

Closes the bindery
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWCloseBindery (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWCloseBindery
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn
(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

NWCloseBindery allows the SUPERVISOR to close and unlock the bindery. The bindery can then be archived.

NWCloseBindery is not supported on NetWare 4.x since 4.x only supports bindery emulation. Direct access of the bindery is not possible on NetWare 4.x. and above.

Because the bindery files contain all the information about the NetWare clients for a server, the bindery should be archived on a regular basis. For bindery files to be archived, the bindery must be closed by calling **NWCloseBindery** since the NetWare server keeps bindery files opened and locked at all times so that they cannot be accessed directly.

After the bindery files have been archived, calling the **NWOpenBindery** function returns control of the bindery files to the NetWare server. While the bindery is closed, much of the functionality of the network is disabled.

NCP Calls

0x2222 23 68	Close Bindery

See Also

NWOpenBindery

NWCreateObject

Creates a bindery object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWCreateObject (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   nuint8          objFlags,
   nuint8          objSecurity);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWCreateObject
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   objFlags : nuint8;
   objSecurity : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the string containing the new object name.

objType

(IN) Specifies the bindery type of the new object.

objFlags

(IN) Specifies whether the new object is dynamic:

BF_DYNAMIC

BF_DYNAMIC
BF_STATIC

objSecurity

(IN) Specifies the access rights mask of the new object.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 E7	E_NO_MORE_USERS
0x89 EE	OBJECT_ALREADY_EXISTS
0x89 EF	INVALID_NAME
0x89 F1	INVALID_BINDERY_SECURITY
0x89 F5	NO_OBJECT_CREATE_PRIVILEGE
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

NWCreateObject requires SUPERVISOR or equivalent rights.

The *objName* and *objType* parameters must uniquely identify the bindery object and cannot contain wildcards.

The bindery object must have a PASSNWOBJ_TYPE to log in to a NetWare server. PASSNWOBJ_TYPE is created by calling the **NWChangeObjectPassword** function.

See Security Rights Mask Values.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 50   Create Bindery Object

See Also

NWChangeObjectPassword, **NWCreateProperty**

NWCreateProperty

Adds a property to a bindery object on the NetWare server associated with the given connection handle

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWCreateProperty (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          propertyName,
   nuint8          propertyFlags,
   nuint8          propertySecurity);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWCreateProperty
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   propertyName : pnstr8;
   propertyFlags : nuint8;
   propertySecurity : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the object name receiving the new property.

objType

(IN) Specifies the type of the affected bindery object.

propertyName

(IN) Points to the name of the property being created.

propertyFlags

(IN) Specifies the bindery flags of the new property (ORed with BF_ITEM or BF_SET):

BF_DYNAMIC
BF_STATIC

propertySecurity

(IN) Specifies the security access mask of the new property.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 ED	PROPERTY_ALREADY_EXISTS
0x89 EF	INVALID_NAME
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F1	INVALID_BINDERY_SECURITY
0x89 F2	NO_OBJECT_READ_PRIVILEGE
0x89 F6	NO_PROPERTY_DELETE_PRIVILEGE
0x89 F7	NO_PROPERTY_CREATE_PRIVILEGE
0x89 FB	N0_SUCH_PROPERTY
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

NWCreateProperty requires Write access to the bindery object.

The requesting process cannot create properties having a greater security level than the access level of the process.

The PASSNWOBJ_TYPE property is created by calling the **NWChangeObjectPassword** function, rather than by calling **NWCreateProperty**.

See Security Rights Mask Values.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 57	Create Property

See Also

NWChangeObjectPassword, **NWCreateObject**

NWDeleteObject

Deletes a bindery object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWDeleteObject (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWDeleteObject
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the object name being deleted.

objType

(IN) Specifies the type of the object being deleted.

Return Values

These are common return values. See Return Values.

0x00	SUCCESSFUL

00	
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F4	NO_OBJECT_DELETE_PRIVILEGE
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

NWDeleteObject requires SUPERVISOR or equivalent rights.

The *objName* and *objType* parameters must uniquely identify the bindery object and cannot contain wildcard characters.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 51	Delete Bindery Object

See Also

NWDeleteObjectFromSet

NWDeleteObjectFromSet

Deletes a member from a bindery property of type SET on the NetWare server associated with the given connection handle

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWDeleteObjectFromSet (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          propertyName,
   pnstr8          memberName,
   nuint16         memberType);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWDeleteObjectFromSet
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   propertyName : pnstr8;
   memberName : pnstr8;
   memberType : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the name of the bindery object whose set is being affected.

objType

(IN) Specifies the object type of the bindery object whose set is being affected.

propertyName

(IN) Points to the name of the property (of type SET) from which the member is being deleted.

memberName

(IN) Points to the name of the bindery object being deleted from the set.

memberType

(IN) Specifies the object type of the member being deleted.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 EA	NO_SUCH_MEMBER
0x89 EB	NOT_GROUP_PROPERTY
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F4	NO_OBJECT_DELETE_PRIVILEGE
0x89 F8	NO_PROPERTY_WRITE_PRIVILEGE
0x89 FB	N0_SUCH_PROPERTY
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 66 Delete Bindery Object From Set

See Also

NWDeleteObject, **NWDeleteProperty**

NWDeleteProperty

Removes a property from a bindery object associated with the specified connection handle

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWDeleteProperty (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8           propertyName);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWDeleteProperty
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   propertyName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the name of the object whose property is being deleted.

objType

(IN) Specifies the type of the object whose property is being deleted.

propertyName

(IN) Points to the property name to be deleted.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F1	INVALID_BINDERY_SECURITY
0x89 F6	NO_PROPERTY_DELETE_PRIVILEGE
0x89 FB	NO_SUCH_PROPERTY
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

NWDeleteProperty requires Write access to the bindery object and the property.

The *objName* and *objType* parameters must uniquely identify the bindery object and cannot contain wildcard characters.

All matching properties of the bindery object are deleted when the *propertyName* parameter contains wildcard characters.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 58	Delete Property

See Also

NWDeleteObjectFromSet, **NWDeleteObject**

NWDisallowObjectPassword

Prevents use of the specified password by the specified object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE NWAPI NWDisallowObjectPassword (
   NWCONN_HANDLE   conn,
   pnstr8         *objName,
   nuint16         objType,
   pnstr8         *disallowedPassword);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWDisallowObjectPassword
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   disallowedPassword : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the name of the object whose password is being disallowed.

objType

(IN) Specifies the type of the object whose password is being disallowed.

disallowedPassword

(IN) Points to the password that is being disallowed.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x00 FF	BINDERY_FAILURE
0x88 01	INVALID_CONNECTION
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 FB	INVALID_PARAMETERS
0x89 FC	NO_SUCH_OBJECT
0x89 FF	HARDWARE_FAILURE

Remarks

The *objName* and *objType* parameters must be specific and cannot contain wildcards.

For **NWDisallowObjectPassword** to work properly, LOGIN_CONTROL must be set appropriately.

NWDisallowObjectPassword adds an encrypted password to the list of old passwords maintained in the OLD_PASSWORDS property. If the OLD_PASSWORDS property does not exist, **NWDisallowObjectPassword** will check UNIQUE_PASSWORDS of the restriction flags in the LOGIN_CONTROL property. If the UNIQUE_PASSWORDS is set, the OLD_PASSWORDS property will be created. Otherwise, a BINDERY_FAILURE error code will be returned.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 53	Get Bindery Object ID
0x2222 23 57	Create Property
0x2222 23 61	Read Property Value
0x2222 23 62	Write Property Value

See Also

NWChangeObjectPassword, NWLoginToFileServer

NWGetBinderyAccessLevel

Returns the access level of the current logged-in entity based on the specified connection handle

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWGetBinderyAccessLevel (
   NWCONN_HANDLE   conn,
   pnuint8         accessLevel,
   pnuint32        objID);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWGetBinderyAccessLevel
  (conn : NWCONN_HANDLE;
   accessLevel : pnuint8;
   objID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

accessLevel

(OUT) Points to the current security access mask for the given connection (optional).

objID

(OUT) Points to the object ID of the current logged in entity (optional).

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY

Remarks

The access level of a process determines which bindery objects and properties the process can find and manipulate.

See Security Rights Mask Values.

NCP Calls

0x2222 23 70   Get Bindery Access Level

NWGetObjectDiskSpaceLeft

Returns the remaining disk space for a specified object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWGetObjectDiskSpaceLeft (
   NWCONN_HANDLE   conn,
   nuint32         objID,
   pnuint32        systemElapsedTime,
   pnuint32        unusedDiskBlocks,
   pnuint8         restrictionEnforced);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWGetObjectDiskSpaceLeft
  (conn : NWCONN_HANDLE;
   objID : nuint32;
   systemElapsedTime : pnuint32;
   unusedDiskBlocks : pnuint32;
   restrictionEnforced : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

(IN) Specifies the ID of the object in question.

systemElapsedTime

(OUT) Points to the time the NetWare server has been up.

unusedDiskBlocks

(OUT) Points to the number of blocks the NetWare server must allocate to a bindery object.

restrictionEnforced

(OUT) Points to a flag indicating whether the NetWare server operating system can limit disk resources:

0x0000	Enforced
0x00FF	Not enforced

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 C6	NO_CONSOLE_PRIVILEGES
0x89 FC	NO_SUCH_OBJECT

Remarks

NWGetObjectDiskSpaceLeft returns the *systemElapsedTime* parameter in approximately 1/18 second units and determines the amount of elapsed time between consecutive calls. When the *systemElapsedTime* parameter reaches 0xFFFF, it resets to zero.

NCP Calls

0x2222 23 230 Get Object's Remaining Disk Space

NWGetObjectEffectiveRights

Returns the effective rights of an object in the specified directory or file

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWGetObjectEffectiveRights (
   NWCONN_HANDLE   conn,
   nuint32         objID,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   pnuint16        rightsMask);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWGetObjectEffectiveRights
  (conn : NWCONN_HANDLE;
   objID : nuint32;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   rightsMask : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

(IN) Specifies the ID of the object in the specified directory or file.

dirHandle

(IN) Specifies the NetWare directory handle associated with the directory path for which the effective rights are desired.

path

(IN) Points to the absolute path (or a path relative to the *dirhandle* parameter) of the directory or file whose effective rights mask is being reported.

rightsMask

(OUT) Points to the rights mask.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 98	VOLUME_DOES_NOT_EXIST
0x89 9B	BAD_DIRECTORY_HANDLE
0x89 9C	INVALID_PATH
0x89 A8	ERR_ACCESS_DENIED
0x89 FB	INVALID_PARAMETERS
0x89 FC	NO_SUCH_OBJECT

Remarks

To determine the effective rights of the requesting workstation, **NWGetObjectEffectiveRights** performs a logical AND between the maximum rights mask of the directory and the current trustee rights of the workstation.

The current trustee rights of the workstation are obtained by performing a logical OR between a trustee access mask of the workstation and the trustee access mask of any object to which the process is security equivalent. The current trustee rights of the workstation may be explicitly listed in the directory or inherited from the parent directory. The maximum rights masks of parent directories do not affect inherited trustee rights.

The *rightsMask* parameter returned to the client indicates which of the eight possible directory rights the client has in the targeted directory. If the *rightsMask* parameter is zero, the client has no rights in the target directory.

See Maximum Rights Mask Values.

NCP Calls

0x2222 22 50 Get Object Effective Rights For Directory Entry

See Also

NWGetEffectiveRights

NWGetObjectID

Looks up an object ID in the bindery on the network server associated with the given connection handle

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWGetObjectID (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnuint32        objID);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWGetObjectID
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   objID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the name of the object in the search.

objType

(IN) Specifies the type of the object in the search.

objID

(OUT) Points to the ID of the found object.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F1	INVALID_BINDERY_SECURITY
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

Since each network server contains its own bindery, object IDs are not consistent across network servers.

The *objName* and *objType* parameters must uniquely identify the bindery object and cannot contain wildcard characters.

The requesting process must be logged in to the network server and have Read access to the bindery object for **NWGetObjectID** to be successful.

NWGetObjectID can be called even if a connection is not authenticated.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 53	Get Bindery Object ID

See Also

NWChangeObjectSecurity, **NWCreateObject**

NWGetObjectName

Returns the name and object type of a bindery object on the network server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWGetObjectName (
   NWCONN_HANDLE   conn,
   nuint32         objID,
   pnstr8          objName,
   pnuint16        objType);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWGetObjectName
  (conn : NWCONN_HANDLE;
   objID : nuint32;
   objName : pnstr8;
   objType : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

(IN) Specifies the object ID.

objName

(OUT) Points to the object name (minimum buffer size=48).

objType

(OUT) Points to the object type (optional).

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 F1	INVALID_BINDERY_SECURITY
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

For **NWGetObjectName** to be successful, the requesting process must be logged in to the network server and have Read access to the bindery object.

All parameter positions must be filled.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 54	Get Bindery Object Name

See Also

NWChangeObjectSecurity, **NWCreateObject**, **NWGetObjectID**

NWIsObjectInSet

Searches a property of type SET for a specified object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWIsObjectInSet (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          propertyName,
   pnstr8          memberName,
   nuint16         memberType);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWIsObjectInSet
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   propertyName : pnstr8;
   memberName : pnstr8;
   memberType : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the name of the object containing the property being searched.

objType

(IN) Specifies the type of the object containing the property being searched.

propertyName

(IN) Points to the property name of the set being searched.

memberName

(IN) Points to the name of the bindery object being searched.

memberType

(IN) Specifies the bindery type of the member being searched.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 EA	NO_SUCH_MEMBER
0x89 EB	NOT_GROUP_PROPERTY
0x89 EC	NO_SUCH_SEGMENT
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 FB	N0_SUCH_PROPERTY

Remarks

NWIsObjectInSet requires Read access to the SET property.

The *objName*, *objType*, *propertyName*, *memberName*, and *memberType* parameters must uniquely identify the property and cannot contain wildcard characters.

NWIsObjectInSet does not expand members of type GROUP in an attempt to locate a specific member; objects must be explicitly in the group. For example, assume the following bindery objects and properties exist:

Object	**Property**	**Property Value**

JOAN		
SECRETA RIES	GROUP_MEM BERS	The object ID of JOAN
EMPLOYE ES	GROUP_MEM BERS	The object ID of SECRETARIES

JOAN is not considered a member of EMPLOYEES; she is not explicitly listed in GROUP_MEMBERS of EMPLOYEES. The bindery does not check for recursive (direct or indirect) membership definitions.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 67 Is Bindery Object In Set

See Also

NWAddObjectToSet, **NWDeleteObjectFromSet**

NWOpenBindery

Reopens a NetWare server bindery closed by calling the **NWCloseBindery** function

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWOpenBindery (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWOpenBindery
   (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 FF	HARDWARE_FAILURE

Remarks

NWOpenBindery is not supported on NetWare 4.x since 4.x only supports bindery emulation. Direct access of the bindery is not possible on NetWare 4.x and above.

The bindery files are normally kept open and locked. Calling **NWOpenBindery** is required only after calling the **NWCloseBindery** function.

Only SUPERVISOR or a bindery object with SUPERVISOR security equivalence can open the bindery.

NCP Calls

0x2222 23 69	Open Bindery

NWReadPropertyValue

Reads the property value of a bindery object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWReadPropertyValue (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          propertyName,
   nuint8          segmentNum,
   pnuint8         segmentData,
   pnuint8         moreSegments,
   pnuint8         flags);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWReadPropertyValue
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   propertyName : pnstr8;
   segmentNum : nuint8;
   segmentData : pnuint8;
   moreSegments : pnuint8;
   flags : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the object name containing the property.

objType

(IN) Specifies the type of the object containing the property.

propertyName

(IN) Points to the name of the property whose information is being retrieved.

segmentNum

(IN) Specifies the segment number of the data (128-byte blocks) to be read (set to 1 initially).

segmentData

(OUT) Points to the 128-byte buffer receiving the property data.

moreSegments

(OUT) Points to a flag indicating if there are more segments to be returned:

0x00	No more segments to be read
0xFF	More segments to be read

flags

(OUT) Points to the property type (optional):

BF_ITEM
BF_SET

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 88	INVALID_FILE_HANDLE
0x89 93	NO_READ_PRIVILEGES
0x89 96	SERVER_OUT_OF_MEMORY
0x89 EC	NO_SUCH_SEGMENT
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F1	INVALID_BINDERY_SECURITY
0x89	NO_PROPERTY_READ_PRIVILEGES

F9	
0x89 FB	N0_SUCH_PROPERTY
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

Read access to the property is required to successfully call **NWReadPropertyValue**.

Each subsequent call to **NWReadPropertyValue** will increment the *segmentNum* parameter until the *moreSegments* parameter is set to 0 or until NO_SUCH_SEGMENT is returned.

The *objName*, *objType*, and *propertyName* parameters must uniquely identify the property and cannot contain wildcard characters.

If the property is of type SET, the data returned in the *segmentData* parameter is an array of bindery object IDs. The bindery attaches no significance to the contents of a property value if the property is of type ITEM.

See Object Type Bindery Parameter.

See Activity Coordination.

NCP Calls

0x2222 23 61 Read Property Value

See Also

NWWritePropertyValue

NWRenameObject

Renames an object in the bindery on the server associated with the connection handle

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWRenameObject (
   NWCONN_HANDLE   conn,
   pnstr8          oldObjName,
   pnstr8          newObjName,
   nuint16         objType);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWRenameObject
  (conn : NWCONN_HANDLE;
   oldObjName : pnstr8;
   newObjName : pnstr8;
   objType : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

oldObjName

(IN) Points to the name of the currently defined object in the bindery.

newObjName

(IN) Points to the new object name.

objType

(IN) Specifies the type of the object.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 EE	OBJECT_ALREADY_EXISTS
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F3	NO_OBJECT_RENAME_PRIVILEGE
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

The *oldObjName, newObjName,* and *objType* parameters must uniquely identify the bindery object and cannot contain wildcard characters. WILD_CARD_NOT_ALLOWED will be returned if the name field strings are not recognized.

Only SUPERVISOR or a bindery object security equivalent to SUPERVISOR can rename bindery objects.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 52	Rename Object

See Also

NWScanObject

NWScanObject

Searches for a bindery object name
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWScanObject (
   NWCONN_HANDLE   conn,
   pnstr8          searchName,
   nuint16         searchType,
   pnuint32        objID,
   pnstr8          objName,
   pnuint16        objType,
   pnuint8         hasPropertiesFlag,
   pnuint8         objFlags,
   pnuint8         objSecurity);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWScanObject
  (conn : NWCONN_HANDLE;
   searchName : pnstr8;
   searchType : nuint16;
   objID : pnuint32;
   objName : pnstr8;
   objType : pnuint16;
   hasPropertiesFlag : pnuint8;
   objFlags : pnuint8;
   objSecurity : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

searchName

(IN) Points to the object name for which to search (wildcards are allowed).

searchType

(IN) Specifies the object type used in the search (wildcards are allowed).

objID

(OUT) Points to the last object ID (-1 is assumed if no value is specified).

objName

(OUT) Points to the name of the next matching object (optional).

objType

(OUT) Points to the 2-byte type of the next matching object (optional).

hasPropertiesFlag

(OUT) Points to the properties flag (optional):

0x00	Matching object has no properties
0xFF	Matching object has properties

objFlags

(OUT) Points to the object flag byte (optional):

BF_STATIC
BF_DYNAMIC

objSecurity

(OUT) Points to the security mask of the matching object (optional).

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 EF	INVALID_NAME
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89	HARDWARE_FAILURE

FF	

Remarks

NWScanObject iteratively scans the bindery for all objects matching both the *objName* and *objType* parameters.

Upon return, the *objID* parameter receives a number to be used as the object identification for the next call.

The requesting process must be logged in to the NetWare server and have Read access to the bindery object.

All parameter positions must be filled.

See Security Rights Mask Values.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 55 Scan Bindery Object

NWScanObjectTrusteePaths

Returns the directory paths to which an object has trustee rights
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWScanObjectTrusteePaths (
   NWCONN_HANDLE   conn,
   nuint32         objID,
   nuint16         volNum,
   pnuint16        iterHandle,
   pnuint8         accessRights,
   pnstr8          dirPath);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWScanObjectTrusteePaths
  (conn : NWCONN_HANDLE;
   objID : nuint32;
   volNum : nuint16;
   iterHandle : pnuint16;
   accessRights : pnuint8;
   dirPath : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

(IN) Specifies the object ID of the user or group for which the trustee information is to be found.

volNum

(IN) Specifies the volume number of the volume being searched.

iterHandle

(IN) Points to the sequence number (set to -1 initially).

accessRights

(OUT) Points to the access mask of the trustee.

dirPath

(OUT) Points to the directory DOS path name of the current trustee (should be at least 270 bytes).

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 98	VOLUME_DOES_NOT_EXIST
0x89 9C	NO_MORE_TRUSTEES
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F2	NO_OBJECT_READ_PRIVILEGE
0x89 FC	NO_SUCH_OBJECT

Remarks

NWScanObjectTrusteePaths iteratively determines all of the directory paths of the bindery object trustee and corresponding access masks.

Upon return, the *iterHandle* parameter is automatically incremented to point to the next directory path. When all valid directory paths have been returned, SUCCESS is returned and the first character of the *dirPath* parameter is set to zero.

To use the DOS path returned by the *dirPath* parameter in subsequent calls, you might have to convert the DOS path to the default name space compatible path.

Only SUPERVISOR, the object, or a bindery object with SUPERVISOR security equivalence can scan the directory paths of an object trustee.

NWScanObjectTrusteePaths was originally written for the 2.x platform and does not handle 3.x and 4.x rights perfectly. For example, **NWScanObjectTrusteePaths** does not return the 2.x "Supervisory" right. To retrieve the correct trustee rights on the 3.x and 4.x platforms, call **NWScanObjectTrusteePaths** to obtain a path. Then call the **NWIntScanForTrustees** function to return the rights of the object to the path.

See Maximum Rights Mask Values.

NCP Calls

0x2222 23 71	Scan Bindery Object Trustee Paths

See Also

NWIntScanForTrustees

NWScanProperty

Scans the given bindery object for properties matching the property name
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWScanProperty (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          searchPropertyName,
   pnuint32        iterHandle,
   pnstr8          propertyName,
   pnuint8         propertyFlags,
   pnuint8         propertySecurity,
   pnuint8         valueAvailable,
   pnuint8         moreFlag);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWScanProperty
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   searchPropertyName : pnstr8;
   iterHandle : pnuint32;
   propertyName : pnstr8;
   propertyFlags : pnuint8;
   propertySecurity : pnuint8;
   valueAvailable : pnuint8;
   moreFlag : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the object name of the set.

objType

(IN) Specifies the object type of the set.

searchPropertyName

(IN) Points to the property name for which to search (wildcards are allowed).

iterHandle

(OUT) Points to the iteration handle to use when making repeated calls (if not specified, -1 is assumed).

propertyName

(OUT) Points to the name of the next matching property (up to 15 characters including the NULL terminator or NULL).

propertyFlags

(OUT) Points to the status flag (optional):

0x00	BF_STATIC
0x00	BF_ITEM
0x01	BF_DYNAMIC
0x02	BF_SET

propertySecurity

(OUT) Points to the security mask (optional).

valueAvailable

(OUT) Points to a flag indicating whether the property has value (optional):

0x00	Property has no value
0xFF	Property has value

moreFlag

(OUT) Points to the more properties flag (optional):

0x00	No more properties exist for object
0xFF	More properties exist

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION

0x89 96	SERVER_OUT_OF_MEMORY
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 FB	NO_SUCH_PROPERTY
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

Upon return, the *moreFlag* parameter contains 0xFF if the matched property is not the last property, and the *iterHandle* parameter contains the number to use in the next call.

NWScanProperty requires Read access to the bindery object as well as the property.

The *objName* and *objType* parameters must uniquely identify the bindery object and cannot contain wildcard characters.

For parameters not desired in the return, NULL can be substituted. All parameter positions must be filled.

See Security Rights Mask Values.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 60 Scan Property

See Also

NWReadPropertyValue, **NWWritePropertyValue**

NWVerifyObjectPassword

Verifies the password of a bindery object on the specified NetWare server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWVerifyObjectPassword (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          password);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWVerifyObjectPassword
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   password : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle whose password is to be verified.

objName

(IN) Points to the name of the object whose password is to be verified.

objType

(IN) Specifies the type of object.

password

(IN) Points to the password to be verified.

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 FB	INVALID_PARAMETERS
0x89 FC	NO_SUCH_OBJECT
0x89 FF	NO_SUCH_OBJECT_OR_BAD_PASSWORD

Remarks

The requesting workstation does not have to be logged in to the NetWare server to call **NWVerifyObjectPassword**.

If the server supports encrypted passwords, the password is encrypted. If the server does not support encryption, password verification is attempted without encryption.

The *objName* and *objType* parameters must uniquely identify the bindery object and cannot contain wildcards.

A bindery object without a PASSWORD is different from a bindery object with a PASSWORD having no value. A workstation is not allowed to log in to a NetWare server as a bindery object that does not have a PASSWORD. A workstation can log in without a password if the bindery object has been given a PASSWORD containing no value.

See Object Type Bindery Parameter.

NCP Calls

0x2222 23 23	Get Login Key
0x2222 23 53	Get Bindery Object ID
0x2222 23 74	Keyed Verify Password

See Also

NWLoginToFileServer

NWWritePropertyValue

Writes the property data of a bindery object on the NetWare server associated with the given connection handle

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Bindery

Syntax

```
#include <nwbindry.h>
or
#include <nwcalls.h>

NWCCODE N_API NWWritePropertyValue (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          propertyName,
   nuint8          segmentNum,
   pnuint8         segmentData,
   nuint8          moreSegments);
```

Pascal Syntax

```
#include <nwbindry.inc>

Function NWWritePropertyValue
  (conn : NWCONN_HANDLE;
   objName : pnstr8;
   objType : nuint16;
   propertyName : pnstr8;
   segmentNum : nuint8;
   segmentData : pnuint8;
   moreSegments : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the name of the object.

objType

(IN) Specifies the type of the object.

propertyName

(IN) Points to the property name of the object.

segmentNum

(IN) Specifies the segment number of the 128-byte data blocks (set to 1 initially).

segmentData

(IN) Points to the 128-byte buffer containing the data.

moreSegments

(IN) Specifies whether more segments are being written:

0xFF	More segments are being written
0x00	The last segment is being written

Return Values

These are common return values. See Return Values.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 E8	WRITE_PROPERTY_TO_GROUP
0x89 EC	NO_SUCH_SEGMENT
0x89 F0	WILD_CARD_NOT_ALLOWED
0x89 F8	NO_PROPERTY_WRITE_PRIVILEGE
0x89 FB	N0_SUCH_PROPERTY
0x89 FC	NO_SUCH_OBJECT
0x89 FE	BINDERY_LOCKED
0x89 FF	HARDWARE_FAILURE

Remarks

A client must have Write access to the property to call **NWWritePropertyValue**.

When **NWWritePropertyValue** returns, any remaining segments are truncated and the extra segments discarded.

Create property value segments sequentially. Before segment N can be created, all segments from 1 to N-1 must be created. However, once all segments of a property value have been established, segments can be written at random. If the segment data is longer than 128 bytes, it is truncated and the 128th byte is NULL.

NOTE: Keep property values to a single segment (128 bytes) to improve bindery efficiency.

The *objName*, *objType*, and *propertyName* parameters must uniquely identify the property and cannot contain wildcard characters.

See Object Type Bindery Parameter.

See Activity Coordination.

NCP Calls

0x2222 23 62   Write Property Value

See Also

NWReadPropertyValue

NDS

NDS: Guides

Introduction to NDS Development: Guide

NDS Architecture: Concepts Guide

NDS Context: Guides

NDS Access: Guides

NDS Partition: Guides

NDS Schema: Guides

NDS Security: Concepts Guide

Introduction to NDS Development: Guide

The information in the following topics are provided for the benefit of developers who are new to NDS programming and those who want a non-technical overview of the subject.

NDS Introduction

Types of Information Stored in NDS

- Names Stored in NDS
- Addresses Stored in NDS
- Lists Stored in NDS
- Descriptions Stored in NDS

Retrieval of Information from NDS

- Lookup in NDS
- Browsing NDS
- Searching NDS

Security of Information in NDS

- Security through NDS Authentication
- Security through Access Control

Administration of Information in NDS

Administration of Information in NDS

NDS Compliance to X.500 Standard

Benefits of NDS

Developing in a Loosely Consistent Environment

Advantages of Loose Consistency

Disappearing NDS Objects

Another Cause of Disappearing NDS Objects

NDS Functions and Structures

NDS: Functions

NDS: Structures

NDS Schema Definitions

NDS Object Class Definitions

Graphical View of NDS Object Class Definitions

NDS Attribute Type Definitions

NDS Attribute Syntax Definitions

NDS Architecture: Concepts Guide

The following topics provide information about NDS architecture:

Architecture Concepts

NDS Architecture Introduction

NDS Workstation Components

Workstation Applications for NDS

Application Programming Interfaces for NDS

Shell Components for NDS

NDS Server Components

Table of NDS Server Utilities

NDS Agents

NDS Bindery

Directory Information Base

Related Topics:

NDS Schema Definitions: Guide

NDS Functions and Structures

NDS: Functions

NDS: Structures

NDS Access: Guides

Information in NDS is organized by objects. An object is a collection of attributes. Each object includes an object class attribute that specifies the other attributes associated with the object.

NDS lets you create and modify objects and read their attributes. The most common types of NDS operations include reading, modifying, and searching for information.

NDS Access lets you authenticate to NDS, read and modify NDS objects, and configure the input and output buffers needed to read from and write to the NDS directory.

The procedures for accessing an object depend to a large extent on the object's class. You must know which attributes are assigned to an object and the associated attribute types and syntaxes. NDS includes a standard set of object classes and attribute types referred to as the NDS Schema.

Before you can access an object, you must first gain access to an NDS tree.

The following topics provide information to help you manage and understand NDS Access.

Guides to NDS Access Tasks

Managing Connections to NDS

Managing Buffers

Managing Objects

Managing Object Attributes

Searching NDS

Guides to NDS Access Concepts

Connection Management

Buffer Management

Object Management

Object Attribute Management

Search Management

NDS Access Functions and Structures

Managing Connections to NDS

Authenticating to NDS

Retrieving Addresses of a Connected Server

Managing Buffers

Preparing NDS Input Buffers

Preparing NDS Output Buffers

Retrieving Results from NDS Output Buffers

Freeing NDS Buffers

Managing Objects

Listing Objects in an NDS Container

Listing Objects in an NDS Container: Example 2

Adding an NDS Object

Adding an NDS Object: Example

Modifying an NDS Object

Modifying an NDS Object: Example

Deleting an NDS Object

Determining the Effective Rights of an Object

Finding the Host Server of an Object

Managing Object Attributes

Reading Attributes of NDS Objects

Reading Attributes of NDS Objects: Example

Comparing Attribute Values

NDS Searches

Searching NDS

Searching for an NDS Object: Example

Connection Management

Authentication

Authentication Introduction

Authentication Requirements

Authentication Process

Background Authentication

Authentication of Client Applications

Encryption in NDS

Public and Private Key Pairs

Password Changing in NDS

Multiple Tree Functions

Buffer Management

NDS Buffer Management Introduction

Buffer Size in NDS

Initialization Operations for NDS Buffers

DEFAULT_MESSAGE_LEN Constant

Buffer Allocation Types and Related Functions

Structure of Output Buffers

NDS Buffer Allocation and Initialization Functions

NDS Input Buffer Functions

NDS Output Buffer Functions

Object Management

Controlling Iterations

Read Requests

Notes About Adding NDS Objects

Change Types for NWDSPutChange

Determining Access Privileges Required for an Operation

NDS List Functions

NDS Read Functions

Configurable NDS Functions

Object Attribute Management

Controlling Iterations

Attribute Value Comparisons

Search Management

NDS Search Introduction

Buffers Needed for NDS Searches

Expression Filters for NDS Searches

Search Parameters

Search Filters

Relational Operators for NDS Searches

Logical Operators for NDS Searches

Search Expression Trees

NDS Access Functions and Structures

NDS: Functions

NDS: Structures

Multiple Tree Functions

NDS Buffer Allocation and Initialization Functions

NDS Input Buffer Functions

NDS Output Buffer Functions

NDS List Functions

NDS Read Functions

Configurable NDS Functions

Related Topics:

NDS Schema: Guides

NDS Context: Guides

An NDS directory context is a memory structure that explains how and where an object fits within a tree. When the context is created, NDS returns a context handle that points to the memory location of the context structure. Most NDS functions require a context handle as a parameter.

The NDS directory context structure is not exposed, and therefore, you can't manipulate it directly. NDS provides functions that allow you to create, read, and modify directory context structures.

The following guide topics provide information to help you manage and understand NDS context.

Guides to NDS Context Tasks

Managing Context

Guides to NDS Context Concepts

Context Naming

Context Management

Managing Context

NDS Context manages directory context memory structures based on a flexable naming structure. The following topics explain the typical tasks it allows you to perform.

Creating a Context

Reading Context Information

Modifying Context Information

Freeing a Context

Context Handling: Example

Context Naming

NDS Names

Hierarchical Naming

Typeless Names

Canonicalized Names

Default Typing Rule

Name Caching

Name Expansion

Alias Naming

Name Functions

Context Management

NDS Context

Context Key Values

Context Flags

Context Name

Country Name

Attribute Type Abbreviations

Character Encoding

Context Settings

NDS Functions and Structures

NDS: Functions

NDS: Structures

Name Functions

NDS Partition: Guides

An NDS database can be sectioned off into managable components called partitions. Each partition contains a part of the overall NDS database that makes business sense to manage as a group. For example, development might be in one partition and operations in another. Partitions are replicated throughout the network. Through replication, members of one partition can have access to the services offered in other partitions without the constant necessity of dealing with the entire network.

The following topics provide information to help you create and manage partitions. They also help you understand NDS partition by describing the distributed nature of the NDS database and explaining how NDS partitions are replicated throughout the network.

Guides to NDS Partition Tasks

Managing Partitions

Managing Replication

Guides to NDS Partition Concepts

Partition Management

Replication Management

Other Partition Concepts

NDS Partition Functions and Structures

Managing Partitions

Adding a Partition

Listing Partitions

Listing Partitions: Example

Retrieving Partition Information from a Server

Joining Partitions

Splitting Partitions

Removing Partitions

Examples

Synchronizing with NDS Database - part 1: Example

Synchronizing with NDS Database - part 2: Example

Synchronizing with NDS Database - part 3: Example

Managing Replication

Adding a Replica

Changing the Type of a Replica

Removing Replicas

Partition Management

Partition Management Introduction

Partitions

Subordinate Partitions

Partition Information

Partition Class

Mandatory Partition Attributes

Optional Partition Attributes

Replica Attribute

Convergence Attribute

All Up To Attribute

Splitting and Joining Partitions

Partition Synchronization

Partition Functions

Replication Management

Replicas

Replica Information

Replica State

Replica Type

Partition Replication

Replica Synchronization

Replication Functions

Other Partition Concepts

Name Server Information

Tree Walking

Progress Reports

Distribution of Access Control Information

Distribution of Schema Information

NDS Partition Functions and Structures

NDS: Functions

NDS: Structures

Partition Functions

Replication Functions

NDS Schema: Guides

The NDS Schema is a set of rules for adding, deleting, and managing information objects in NDS. Through these rules, the schema sets the pattern of organization for an NDS database. The schema controls not only the structure of individual objects, but also the relationship among objects in the NDS tree.

The following topics provide information to help you understand NDS Schema.

NDS Schema Overview

Introduction to the NDS Schema

NDS Schema Components

NDS Schema Definitions: Guide

Guides to NDS Schema Tasks

Managing Attribute Definitions

Managing Class Definitions

Working with Syntax Definitions

Guides to NDS Schema Concepts

Class Definition Management

Attribute Definition Management

Schema Extensions

Schema Definition Changes

NDS Schema Functions and Structures

NDS Schema Definitions: Guide

Managing Attribute Definitions

Creating an Attribute Definition

Reading an Attribute Definition

Deleting an Attribute Definition

Managing Class Definitions

Creating a Class Definition

Modifying a Class Definition

Reading a Class Definition

Deleting a Class Definition

Listing Containable Classes

Working with Syntax Definitions

Retrieving Syntax Names and Definitions

NDS Functions and Structures

NDS: Functions

NDS: Structures

NDS Schema Definitions: Guide

NDS Object Class Definitions

Graphical View of NDS Object Class Definitions

NDS Attribute Type Definitions

NDS Attribute Syntax Definitions

Class Definition Management

Object Classes in NDS

Object Class Rules

Structure Rules

Containment Classes

Contained By Classes

Name Rules

Super Class Structure

Inheritance

Class Inheritance Rules

Effective and Noneffective Classes

Class Definition Creation

Class Maintenance Information

Restrictions on New Classes

NDS Class Functions

Attribute Definition Management

Attribute Types and Attribute Syntax

Attribute Syntax

Attribute Constraints

Mandatory and Optional Attributes

Attribute Type Information

Attributes and Classes

NDS Attribute Functions

Schema Extensions

NDS Schema Extensions

Registering Attribute Types and Class Definitions

Distribution of the NDS Schema

Schema Definition Changes

Schema Changes for NetWare 4.01

Schema Changes for NetWare 4.1

NDS Schema Functions and Structures

NDS: Functions

NDS: Structures

NDS Class Functions

NDS Attribute Functions

NDS Security: Concepts Guide

NDS Security controls access to objects in NDS and their attributes. The following topics provide information to help you understand *NDS Security*.

Introduction to NDS Security

NDS Security Introduction

Access Control Lists and Inheritance

Access Control Lists

Default ACL Templates

Inheritance in NDS

Inheritance Masks

NDS Privileges

Rights to an NDS Object

Access Privileges

Rights to the Properties of an NDS Object

Giving Rights in NDS

Effective Rights Calculations

Other NDS Security Topics

Trustees

Equivalence in NDS

NDS in Security Applications

Object Management in NDS

NDS Access and the File System

NDS Security Function

NDS Functions and Structures

NDS: Functions

NDS: Structures

NDS Security Function

NDS: Tasks

Adding a Partition

Follow this procedure to create a new partition along with its root object.

1. **Allocate the request buffer by calling NWDSAllocBuf.**
2. **Initialize the request buffer for an DSV_ADD_PARTITION operation by calling NWDSInitBuf.**
3. **Place the name of one of the object's attributes in the request buffer by calling NWDSPutAttrName.**
4. **Place a value for the attribute (in step 3) in the buffer by calling NWDSPutAttrVal.**
5. **Loop to step 3 until the names and values for all of the desired attributes have been placed in the buffer.**
6. **Add the new partition by calling NWDSSplitPartition.**
7. **Free the request buffer when it is no longer needed by calling NWDSFreeBuf.**

Since each partition contains a root object, **NWDSSplitPartition** also creates the root object for the new partition. You must provide the name and attribute values for the new object. In addition to any attributes required by the object's base class, supply a Convergence value for the partition. All other partition attributes are assigned values automatically. The initial replica attribute for a new partition is always type master replica.

Related Topics:

Listing Partitions

Partition Information

NDS Partition: Guides

Adding a Replica

Add a replica of an existing partition to a server by calling **NWDSAddReplica**.

This function only adds replicas of type secondary or read-only. You

This function only adds replicas of type secondary or read-only. You identify the partition to be replicated by supplying the name of the partition's root object.

Related Topics:

Changing the Type of a Replica

NDS Partition: Guides

Adding an NDS Object

To add an object to NDS, follow these steps:

1. **Allocate memory for the request buffer by calling NWDSAllocBuf.**
2. **Set the *iterationHandle* to NO_MORE_ITERATIONS.**
3. **Initialize the request buffer for a DSV_ADD_ENTRY (7) operation by calling NWDSInitBuf.**
4. **For each attribute to be supplied for the object, first store the attribute's name in the result buffer by calling NWDSPutAttrName. Then store the associated value(s) in the result buffer by calling NWDSPutAttrVal once for each value.**

   You must supply the mandatory values for the object class as dictated by the schema.
5. **Create the new object by calling NWDSAddObject.**
6. **Free the request buffer by calling NWDSFreeBuf.**

   NOTE: The name of the object is specified by *objectName*. The naming attribute is mandatory, but it is speechified in the call. It is does not have to be explicitly placed in the *objectInfo* buffer.

Related Topics:

Adding an NDS Object: Example

Notes About Adding NDS Objects

NDS Access: Guides

Authenticating to NDS

Most client workstations log in to the network when they are booted making it unnecessary for many client applications to perform this task. See Authentication of Client Applications.

If you want your application to have full responsibility for accessing the network, or if you are writing an NLM™ application that must access NDS or another NLM on a different server, you can control the authentication process by following these steps.

1. **Initialize an NDS context by calling NWDSCreateContextHandle.**

2. **If needed, call NWDSSetContext to change context values.**

   For information about changing your context, see NDS Context and Modifying Context Information.

3. **Log in to NDS by calling NWDSLogin.**

4. **Open a new connection by calling either NWDSOpenConnToNDSServer, NWCCOpenConnByName, or NWCCOpenConnByRef.**

5. **Authenticate and license the new connection by calling NWDSAuthenticate.**

   NOTE: The process of authenticating to NDS is the same for client applications and NLMs. The only difference is that NLMs do not inherit the credentials of the computer they are running on.

   Although an NLM has administrator rights to the local file system directory, it is not authenticated to NDS as "admin"; it is authenticated as "public". If you want to do anything with NDS other than read public information, you must log in. The authentication credentials are stored on the thread group level and are only accessible by the OS.

Related Topics:

Retrieving Addresses of a Connected Server

NDS Access: Guides

Changing the Type of a Replica

Change a replica's type by calling **NWDSChangeReplicaType**.

You identify the replica by passing in the name of the server containing the replica and the name of the partition's root object. If you change a replica's type to master, the current master replica will be changed to a secondary replica.

Related Topics:

Removing Replicas

NDS Partition: Guides

Comparing Attribute Values

Comparing Attribute Values

Follow these steps to compare an object's attribute value with another value:

1. **Allocate a request buffer by calling NWDSAllocBuf.**
2. **Initialize the request buffer for a DSV_COMPARE operation by calling NWDSInitBuf.**
3. **Place the name of the attribute whose value you want to compare into the request buffer by calling NWDSPutAttrName.**
4. **Place the value you want to compare into the buffer by calling NWDSPutAttrVal.**
5. **Compare the values by calling NWDSCompare.**
6. **Check *matched* to see if the values matched.**
7. **Free the request buffer by calling NWDSFreeBuf.**

Related Topics:

Searching NDS

NDS Access: Guides

Creating a Class Definition

Defining a new object class for the NDS Schema is done with the following steps:

1. **Allocate a request buffer by calling NWDSAllocBuf.**
2. **Initialize the request buffer for a DSV_DEFINE_CLASS operation by calling NWDSInitBuf.**
3. **Prepare the request buffer for storing object-class names in the Super Class Names list by calling NWDSBeginClassItem.**
4. **Place the desired object-class names in the Super Class List by calling NWDSPutClassItem once for each object-class name to be placed in the list.**
5. **Prepare the request buffer for storing object-class names in the Containment Class Names list by calling NWDSBeginClassItem.**
6. **Place the desired object-class names in the Containment Class Names list by calling NWDSPutClassItem once for each object-class name to be placed in the list.**
7. **Prepare the request buffer for storing naming-attribute names in the Naming Attributes List by calling NWDSBeginClassItem.**

8. **Place the desired naming-attribute names in the Naming Attributes List by calling NWDSPutClassItem once for each naming-attribute name to be placed in the list.**

9. **Prepare the request buffer for storing attribute names in the Mandatory Attribute Names list by calling NWDSBeginClassItem.**

10. **Place the desired attribute names in the Mandatory Attribute Names list by calling NWDSPutClassItem once for each attribute name to be placed in the list.**

11. **Prepare the request buffer for storing attribute names in the Optional Attribute Names list by calling NWDSBeginClassItem.**

12. **Place the desired attribute names in the Optional Attribute Names list by calling NWDSPutClassItem once for each attribute name to be added to the list.**

13. **Add the object-class definition to the NDS Schema by calling NWDSDefineClass.**

14. **Free the request buffer by calling NWDSFreeBuf.**

    NOTE: If you do not have any object names or attribute names you want to add to one of the lists, you must still call **NWDSBeginClassItem** to move to the list. You then immediately call **NWDSBeginClassItem** again to move to the next list.

Related Topics:

Creating a Class Definition: Example

NDS Schema: Guides

Creating a Context

Call **NWDSCreateContextHandle** to create a context as follows:

```
NWDSCCODE           ccode;NWDSContextHandle    context; ccode = NWDSCr

     /* handle creation error */
```

Related Topics:

Reading Context Information

Modifying Context Information

Freeing a Context

Context Handling: Example

NDS Context: Guides

Creating an Attribute Definition

1. **Declare a structure of type NWATTR_INFO and fill it in.**

   For example, suppose you wanted to create an attribute called "Toast Setting". You would declare a structure and fill it in, as follows:

   ```
   NWDSCCODE     ccode;                        // Declare new attribute
   NWATTR_INFO   toastSet;
   structure

   toastSet.attrFlags=DS_SINGLE_VALUED_ATTR   // Constraints set to sin
   value
   toastSet.attr
   SyntaxID=SYN_INTEGER                       // Syntax set to integer
   toastSet.attrLower=-10;                    // Lower value limit set
   toastSet.attrUpper=10;                     // Upper value limit set
   toastSet.asn1ID.length=0;                  // Not used in this examp
   toastSet.asn1ID.data[0]=0;                 // Not used in this examp
   ```

2. **Create a new attribute definition to define the attribute using NWDSDefineAttr, as illustrated in the following example:**

   ```
   ccode = NWDSDefineAttr(context, "Toast Setting", &toastSet);
   if (ccode != 0)
      printf("Error while defining attribute: d%\n", ccode);
   ```

   NWDSDefineAttr requires three arguments: the directory context handle, the name of the new attribute in a string, and the address of the data structure of the attribute.

Related Topics:

Reading an Attribute Definition

NDS Schema: Guides

Deleting a Class Definition

You can delete a class definition using a straightforward function called **NWDSRemoveClassDef**, as shown in the following example:

```
ccode = NWDSRemoveClassDef(context, "Toaster");
if (ccode != 0)
   printf("Error while deleting class definition: %d\n", ccode);
```

NWDSRemoveClassDef requires two arguments: the NDS context handle and the name of the class to be deleted.

NOTE: You cannot remove Base class definitions, nor can you remove any class until all of the objects associated with that class have been removed from the NDS tree.

Related Topics:

Listing Containable Classes

NDS Schema: Guides

Deleting an Attribute Definition

Delete an attribute definition by calling **NWDSRemoveAttrDef**. If you wanted to delete the attribute "Toast Setting", you would enter the following code:

```
ccode = NWDSRemoveAttrDef(context, "Toast Setting");
if (ccode != 0)
   printf("Error while removing attribute: %d\n", ccode);
```

This function only requires two arguments: the NDS context handle and the name of the attribute to be deleted.

NOTE: You cannot remove an attribute that is part of the Base class, nor can you remove an attribute that you have added to the Base class definition.

Related Topics:

Creating a Class Definition

NDS Schema: Guides

Deleting an NDS Object

There is no need to prepare a buffer when performing this task.

To Delete an Object, call **NWDSRemoveObject**, which requires only the context handle and the name of the object to be removed, as shown in the following example:

```
ccode = NWDSRemoveObject(contextHandle, objectName);
```

NOTE: You cannot remove an object that contains subordinates. All objects in a container must be removed before the container object can be removed.

Related Topics:

Determining the Effective Rights of an Object

Reading Effective Rights: Example

Deleting a User Object: Example

NDS Access: Guides

Determining the Effective Rights of an Object

Determine an object's effective privileges on another object by following these steps:

1. **Allocate the result buffer by calling NWDSAllocBuf. This buffer does not need to be initialized since it is a result buffer.**
2. **If you want to retrieve information for selected attributes, complete steps 3 through 5. To retrieve information for all of the object's attributes, skip to step 6.**
3. **Allocate the request buffer by calling NWDSAllocBuf.**
4. **Initialize the request buffer for a DSV_READ operation by calling NWDSInitBuf.**
5. **Place the attribute names in the request buffer by calling NWDSPutAttrName once for each attribute name.**
6. **Set *interationHandle* to NO_MORE_ITERATIONS.**
7. **Call NWDSListAttrsEffectiveRights.**
8. **Determine the number of attributes in the result buffer by calling NWDSGetAttrCount.**
9. **For each attribute in the result buffer, retrieve the information by calling NWDSGetAttrVal.**
10. **If the iteration handle is not equal to NO_MORE_ITERATIONS, loop to step 7. Otherwise, go to step 11.**
11. **Free the request buffer by calling NWDSFreeBuf.**
12. **Free the result buffer by calling NWDSFreeBuf.**

NDS fills results buffers with discrete units of information. If the whole unit cannot fit into the buffer, the entire unit will be withheld until the next iteration. For **NWDSListAttrsEffectiveRights**, a unit of information consists of an attribute name and privilege.

Aborting Iterative Operations

If you need information on aborting an iterative operation, see Controlling Iterations.

Related Topics:

Finding the Host Server of an Object

NDS Access: Guides

Finding the Host Server of an Object

The steps for determining the addresses on a server where an object is located are as follows:

1. **Allocate a result buffer by calling NWDSAllocBuf. This buffer does not need to be initialized since it is a result buffer.**
2. **Call NWDSGetObjectHostServerAddress.**
3. **Call NWDSGetAttrCount to determine the number of attributes stored in the result buffer.**
4. **Call NWDSGetAttrName to retrieve the attribute name (network address) and the count of attribute values.**
5. **For each attribute value, call NWDSComputeAttrValSize to find the size of the current address in the result buffer.**
6. **Allocate a block of memory the size of the attribute value.**
7. **Retrieve the current address from the result buffer by calling NWDSGetAttrVal and passing in the pointer allocated in step 5.**
8. **When NWDSGetAttrVal returns, typecast the pointer to be a pointer to Net_Address_T, to access the information.**
9. **Free the allocated memory before retrieving the next address. (Network addresses can be different sizes.)**
10. **Loop to step 4 until all addresses have been removed from the result buffer.**

Related Topics:

Reading Attributes of NDS Objects

NDS Access: Guides

Freeing a Context

Call **NWDSFreeContext** to free the context memory as follows:

```
NWDSCCODE err;
...
// Now free the context before exiting the program.
```

```
err = NWDSFreeContext(context);
if(!err)
   printf("\n\nContext was freed\n");
else
   printf("\n\nError <%d> occurred while freeing context\n", err);
```

Related Topics:

Context Handling: Example

Creating a Context

NDS Context: Guides

Freeing NDS Buffers

This task releases a buffer allocated by **NWDSAllocBuf**. After you have executed an operation using a buffer and retrieved the information from the output buffer (if applicable), always free the buffer memory.

Free buffer memory by calling **NWDSFreeBuf**, as shown in the following example:

```
NWDSFreeBuf(outBuf); // Always returns successful
```

Related Topics:

Listing Objects in an NDS Container

NDS Access: Guides

Joining Partitions

Join a subordinate partition to its parent by calling **NWDSJoinPartitions**.

You can perform this operation only on master replicas residing on the same server. You can join two partitions only if there are no secondary or read-only replicas of either partition. The two partitions must include a subordinate partition and its parent.

Related Topics:

Splitting Partitions

NDS Partition: Guides

Listing Containable Classes

Use this procedure to list the object classes that can be contained by (are subordinate to) a specified object.

1. **Allocate a result buffer by calling NWDSAllocBuf. (The buffer does not need to be initialized since it is a result buffer.)**
2. **Set *iterationHandle* to NO_MORE_ITERARTIONS.**
3. **Retrieve the object classes the parent object can contain by calling NWDSListContainableClasses.**
4. **Determine the number of object-class names contained in the buffer by calling NWDSGetClassItemCount.**
5. **For each object class name in the result buffer, retrieve the name by calling NWDSGetClassItem.**
6. **If the value of the iteration handle is not equal to NO_MORE_ITERATIONS, go to step 3. Otherwise, go to step 7.**
7. **Free the result buffer by calling NWDSFreeBuf.**

Aborting Iterative Operations

If you need information on aborting an iterative operation, see Controlling Iterations.

Related Topics:

Retrieving Syntax Names and Definitions

NDS Schema: Guides

Listing Objects in an NDS Container

This task finds all of the immediate subordinates of an object.

1. **Allocate memory for the output buffer by calling NWDSAllocBuf.**
2. **Set the iteration handle to NO_MORE_ITERATIONS.**
3. **Call NWDSList.**
4. **Determine the number of subordinate objects in the output buffer by calling NWDSGetObjectCount.**
5. **Call NWDSGetObjectName for each subordinate object in the output buffer.**
6. **If the iteration handle is not equal to NO_MORE_ITERATIONS, loop to step 3. Otherwise, go to step 7.**
7. **Free the output buffer by calling NWDSFreeBuf.**

Aborting Iterative Operations

If you need information on aborting an iterative operation, see Controlling Iterations.

Related Topics:

Listing Objects in an NDS Container: Example 2

NDS List Functions

NDS Access: Guides

Listing Partitions

List the partitions stored on a particular server by calling **NWDSListPartitions**.

The results include the partition name and replica type.

Related Topics:

Listing Partitions: Example

NWDSListPartitionsExtInfo

NDS Partition: Guides

Modifying a Class Definition

Optional attributes can be added to an object-class definition by using the following steps:

1. **Allocate a request buffer by calling NWDSAllocBuf.**
2. **Initialize the request buffer for a DS_MODIFY_CLASS_DEF operation by calling NWDSInitBuf.**
3. **For each optional attribute to be added to the class definition, store the attribute's name in the request buffer by calling NWDSPutAttrName.**
4. **Modify the object-class definition by calling NWDSModifyClassDef.**
5. **Free the request buffer by calling NWDSFreeBuf.**

Related Topics:

Reading a Class Definition

NDS Schema: Guides

Modifying an NDS Object

Modifying objects that already exist in NDS is very similar to adding a new object.

1. **Allocate a data buffer by calling NWDSAllocBuf.**

2. **Initialize the buffer for a DSV_MODIFY_ENTRY (9) operation by calling NWDSInitBuf.**

3. **For each attribute value to be modified, place the desired changes into the buffer using NWDSPutChange and NWDSPutAttrVal.**

   NWDSPutChange is used to indicate which attribute is to be modified, and **NWDSPutAttrVal** places the new attribute value into the buffer.

4. **Make the modification by calling NWDSModifyObject.**

The *attrName* parameter simply refers to a text string containing the attribute name (for example "Surname"). The *buf* parameter points to a buffer that has been allocated by calling **NWDSAllocBuf** and initialized by calling **NWDSInitBuf**. Make sure you use the DSV_MODIFY_ENTRY operation when you initialize the buffer. The *changeType* parameter refers to an integer value that defines what type of operation will be made on the named attribute. See Change Types for NWDSPutChange.

Remember, if an attempt is made to modify the Object Class attribute, an error is returned. Also, a value can be modified by placing a combination of DS_REMOVE_VALUE and DS_ADD_VALUE change records in the same request buffer. This allows the operations to be completed by calling **NWDSModifyObject** once.

After putting the change record in the buffer, call **NWDSPutAttrVal** to put the value of the attribute in the buffer.

Related Topics:

Modifying an NDS Object: Example

NDS Access: Guides

Modifying Context Information

Call **NWDSSetContext** to modify context information as follows:

```
NWDSCCODE err;
char newContextName[MAX_DN_CHARS+1]; /* ((MAX_DN_CHARS+1)*2)for unicode
/* change the context name */
printf("\n\nEnter a new name context: \n");
gets(newContextName);
err=NWDSSetContext(context,DCK_NAME_CONTEXT,newContextName);
if(err)
```

```
{
   printf("\n\nNWDSSetContext returned error <%d>",err);
}
else
{
   printf("\n\nNWDSSetContext returned <%d>",err);
   printf("\nName context is set.");
}
```

Related Topics:

Freeing a Context

Reading Context Information

Context Key Values

Context Handling: Example

NDS Context: Guides

Preparing NDS Input Buffers

This task prepares a memory buffer for writing data to the NDS directory.

1. **Allocate memory for the input buffer by calling NWDSAllocBuf. The following code allocates a buffer of size DEFAULT_MESSAGE_LEN Constant.**

```
pBuf_T    inputBuffer;

ccode = NWDSAllocBuf (DEFAULT_MESSASE_LEN, &inputBuffer);
if (ccode)
    printf("Error while allocating buffer: %d\n", ccode);
```

2. **Initialize the input buffer by calling NWDSInitBuf. Choose one of the Initialization Operations for NDS Buffers according to the intended operation. The following code initializes an input buffer for a read operation.**

```
ccode = NWDSInitBuf(context, DSV_READ, &inputBuffer);
if (ccode)
    printf("Error while initializing buffer: %d\n", ccode);
```

3. **Place the input data into the buffer using the NDS Input Buffer Functions that correspond to the data type you are handling.**

```
ccode = NWDSAllocBuf (DEFAULT_MESSAGE_LEN, &inputBuffer);
if (!ccode)
{  ccode = NWDSInitBuf (context, DSV_READ, &inputBuffer);
   ccode = NWDSPutAttrName (context, inputBuffer, "surname");
   ccode = NWDSPutAttrName (context, inputBuffer, "CN");
```

```
        ccode = NWDSPutAttrName (context, inputBuffer, "Login Script");
        ccode = NWDSPutAttrName (context, inputBuffer, "Language");
        ccode = NWDSPutAttrName (context, inputBuffer, "Email Address");

    }
```

NOTE: Don't add data to a buffer directly. NDS contains a complete set of input buffer functions that operate on buffers allowing you to enter data. To add data to the Buffer, you must use the function that corresponds to the data type you are handling.

Related Topics:

Preparing NDS Output Buffers

Freeing NDS Buffers

NDS Access: Guides

Preparing NDS Output Buffers

This task prepares a buffer for retrieving data from an NDS directory.

Allocate memory for the input buffer by calling **NWDSAllocBuf**. The following code allocates a buffer of size DEFAULT_MESSAGE_LEN Constant.

```
pBuf_T    outputBuffer;

ccode = NWDSAllocBuf (DEFAULT_MESSASE_LEN, &outputBuffer);
if (ccode)
    printf("Error while allocating buffer: %d\n", ccode);
```

NOTE: Unlike an input buffer, you don't need to initialize an output buffer.

Don't retrieve or delete data from a buffer directly. NDS contains a complete set of NDS Output Buffer Functions that operate on buffers allowing you to retrieve data. To read data from the Buffer, you must use the function that corresponds to the data type you are handling.

Related Topics:

Retrieving Results from NDS Output Buffers

Preparing NDS Input Buffers

NDS Access: Guides

Reading a Class Definition

If you want to determine the names of all of the classes and their definitions, use **NWDSReadClassDef** as follows:

```
ccode = NWDSReadClassDef(context, DS_CLASS_DEF_NAMES, TRUE, NULL,
                         &iterationHandle, outBuffer);
if (ccode != 0)
   printf("Error while reading class definition: %d\n", ccode);
```

NWDSReadClassDef requires six arguments: the NDS context handle, the information type specifier (names only, or names and definitions), the all-classes flag set, NULL instead of a pointer to an input buffer, the address of the iteration handle, and a pointer to the output buffer.

Read each class entry from the output buffer using **NWDSGetClassDefCount** and **NWDSGetClassDef**, followed by five consecutive calls to **NWDSGetClassItemCount** and **NWDSGetClassItem**.

> **NOTE:** A call to **NWDSReadClassDef** returns only the attributes that were defined for that particular class. It does not return the attributes that were inherited, but it does return the name of the super class. To find all of the attributes available for a class, call **NWDSReadClassDef** for each super class until you reach Top.

Related Topics:

Creating a Class Definition: Example

NDS Schema: Guides

Reading an Attribute Definition

Reading an attribute definition by using **NWDSReadAttrDef** is shown in the following example:

```
ccode = NWDSReadAttrDef(context, DS_ATTR_DEFS, FALSE, inBuf,
                        &iterationHandle, outBuf);
if (ccode != 0)
   printf("Error reading attribute definition: %d\n", ccode);
```

NWDSReadAttrDef requires five arguments: the NDS context handle, the type of information required, the all-attributes flag cleared, a pointer to the request buffer that contains the name of the attribute, the address of the iteration handle, and a pointer to the output buffer that receives the attribute definition.

Related Topics:

Reading a Class Definition: Example

Reading an Attribute Definition: Example

Reading Non-attribute Object Information: Example

NDS Schema: Guides

Reading Attributes of NDS Objects

Read the attributes of an object by following these steps:

1. **Allocate memory for the output buffer by calling NWDSAllocBuf.**
2. **If you are requesting information for all attributes, skip to step 6.**
3. **Call NWDSAllocBuf to allocate memory for the input buffer.**
4. **Call NWDSInitBuf to initialize the input buffer for a DSV_READ operation.**
5. **Call NWDSPutAttrName once for each attribute name being placed in the input buffer.**
6. **Set iteration handle to NO_MORE_ITERATIONS.**
7. **Call NWDSRead.**
8. **Call NWDSGetAttrCount to determine the number of attributes in the output buffer.**
9. **Call NWDSGetAttrName to retrieve the name of the current attribute and the number of its values from the output buffer.**
10. **Determine the size of the attribute value; then call NWDSComputeAttrValSize to allocate a block of memory that size.**
11. **Call NWDSGetAttrVal to read the attribute value. For multivalued attributes, call this function for each of the values.**
12. **Free the memory that was allocated in step 10.**
13. **If all of the attribute information has not been read from the output buffer, loop to step 9.**
14. **If the iteration handle is not equal to NO_MORE_ITERATIONS, loop to step 7.**
15. **If an input buffer was allocated, free the input buffer by calling NWDSFreeBuf.**
16. **Free the output buffer by calling NWDSFreeBuf.**

The results of **NWDSRead** are not ordered and might not appear in alphabetical order.

If *infotype* is set to DS_ATTRIBUTE_VALUES, specifying the Read operation should return both attribute names and values. You must retrieve the information in the correct order; attribute name first, then all of the values associated with the attribute. Then you must retrieve the next attribute name and its values and so on. Otherwise, **NWDSGetAttrName** will return erroneous information.

Aborting Iterative Operations

If you need information on aborting an iterative operation, see Controlling Iterations.

Related Topics:

Reading Attributes of NDS Objects: Example

Read Requests

NDS Read Functions

NDS Access: Guides

Reading Context Information

Call **NWDSGetContext** to read context information as follows:

```
void ShowNameContext(NWDSContextHandle context)
{
   NWDSCCODE err;
   char name[MAX_DN_CHARS+1]; /* ((MAX_DN_CHARS+1)*2)for unicode */
   err = NWDSGetContext(context, DCK_NAME_CONTEXT, name);
   if(err)
   {
      printf("\n\nNWDSSetContext returned error <%d>",err);
   }
   else
   {
      printf("\nCurrent Name Context: %s",name);
   }
}
```

Related Topics:

Modifying Context Information

Freeing a Context

Context Key Values

Context Handling: Example

NDS Context: Guides

Removing Partitions

Delete a partition by calling **NWDSJoinPartitions**.

You can remove a partition only if there are no secondary or read-only replicas of the partition.

Related Topics:

Adding a Replica

NDS Partition: Guides

Removing Replicas

Delete a replica of a partition by calling **NWDSRemoveReplica**.

You identify the replica by supplying the name of the server and the name of the partition's root object. This function can remove only secondary and read-only replicas.

NOTE: **NWDSJoinPartition** can remove the master replica.

Related Topics:

Adding a Partition

Adding a Replica

NDS Partition: Guides

Retrieving Addresses of a Connected Server

To determine the network addresses for a server associated with a connection, follow these steps:

1. **Allocate a result buffer by calling NWDSAllocBuf. This buffer does not need to be initialized since it is a result buffer.**
2. **Call NWDSGetServerAddresses.**
3. **Determine the number of addresses stored in the result buffer by calling NWDSGetAttrCount.**
4. **Call NWDSComputeAttrValSize to find the size of the address data in the buffer.**
5. **Allocate a contiguous block of memory the size of the attribute value, and set a void pointer to point to that block.**
6. **Call NWDSGetAttrVal, passing in the pointer to the allocated memory.**

7. **When NWDSGetAttrVal returns, typecast the pointer to be a pointer to Net_Address_T, and retrieve the information.**

8. **Before retrieving the next address, free the allocated memory. (Addresses can be different sizes.)**

9. **Loop to step 4 until all addresses have been removed from the result buffer.**

10. **Free the result buffer by calling NWDSFreeBuf.**

When all addresses have been retrieved, free the result buffer pointer to *netAddresses*.

Related Topics:

Preparing NDS Input Buffers

NDS Access: Guides

Retrieving Partition Information from a Server

Partition information is retrieved by using the following steps:

1. **Allocate a result buffer to receive the results by calling NWDSAllocBuf. (The buffer does not need to be initialized since it is a result buffer.)**

2. **Set the iteration handle to NO_MORE_ITERATIONS.**

3. **Obtain the partition information by calling NWDSListPartitions.**

4. **Determine the number of partitions whose information is stored in the request buffer by calling NWDSGetServerName.**

5. **For each partition whose information is stored in the buffer, retrieve the partition information by calling NWDSGetPartitionInfo.**

6. **If the iteration handle is set to NO_MORE_ITERATIONS, go to step 7; otherwise, loop to step 3.**

7. **Free the buffer when it is no longer needed by calling NWDSFreeBuf.**

If you decide to stop retrieving partition information before *iterationHandle* is set to NO_MORE_ITERATIONS, call **NWDSCloseIteration** to free memory and state information associated with the partition listing operation.

Related Topics:

Joining Partitions

NDS Access: Guides

Retrieving Results from NDS Output Buffers

This task reads data from an output buffer.

1. **Determine the number of objects in the buffer by calling NWDSGetObjectCount.**

2. **Determine the amount of memory required for each object attribute by calling NWDSComputeAttrValSize.**

3. **Allocate memory to receive the attribute data.**

4. **Retrieve the attribute data from the buffer by calling the NDS Output Buffer Functions that correspond to the data type you are handling.**

5. **Loop through steps 2, 3, and 4 until all attributes of every object in the buffer has been retrieved.**

   NOTE: Data must be read from an output buffer sequentially. Do not skip an item, even if you already know its value.

Related Topics:

Freeing NDS Buffers

NDS Access: Guides

Retrieving Syntax Names and Definitions

To retrieve names and definitions for all syntaxes in the NDS Schema, use the following steps:

1. **Allocate memory for the result buffer by calling NWDSAllocBuf. (This buffer does not need to be initializes, since it is a result buffer.)**

2. **Set the contents of the iteration handle to NO_MORE_ITERATIONS.**

3. **Call NWDSReadSyntaxes with *infoType*=DS_SYNTAX_DEFS, *allSyntaxes*=TRUE, and *syntaxNames*=NULL.**

4. **Determine the number of syntax definitions in the result buffer by calling NWDSGetSyntaxCount.**

5. **For each syntax in the result buffer, retrieve the syntax name and definition by calling NWDSGetSyntaxDef.**

6. **If the contents of the iteration handle is not set to NO_MORE_ITERATIONS, loop to step 3. Otherwise, go to step 7.**

7. **Free the result buffer by calling NWDSFreeBuf.**

If both syntax names and values are stored in the buffer, the names and values must be pulled from the buffer. If both names and values are requested and names only are extracted, erroneous information results.

To retrieve all information about specific syntaxes in the NDS Schema, do the following:

1. **Allocate memory for the request buffer by calling NWDSAllocBuf.**
2. **Initialize the request buffer for a DSV_READ_SYNTAXES operation by calling NWDSInitBuf.**
3. **For each syntax whose information you want to retrieve, store the syntax's name in the request buffer by calling NWDSPutSyntaxName .**
4. **Allocate memory for the result buffer by calling NWDSAllocBuf. (This buffer does not need to be initialized since it is a result buffer.)**
5. **Set the contents of the iteration handle to NO_MORE_ITERATIONS.**
6. **Call NWDSReadSyntaxes with *infoType*=DS_SYNTAX_DEFS, *allSyntaxes*=FALSE, and *syntaxNames*=the address of the request buffer.**
7. **Determine the number of syntax definitions in the result buffer by calling NWDSGetSyntaxCount.**
8. **For each syntax in the buffer, retrieve the syntax name and definition by calling NWDSGetSyntaxDef.**
9. **If the contents of the iteration handle is not set to NO_MORE_ITERATIONS, loop to step 6. Otherwise, go to step 10.**
10. **Free the request buffer by calling NWDSFreeBuf.**
11. **Free the result buffer by calling NWDSFreeBuf.**

Related Topics:

Creating an Attribute Definition

NDS Schema: Guides

Searching NDS Guide

Searching NDS is presented here as a sequential list of subordinate tasks to accomplish.

Follow these steps to search a region of NDS:

1. **Call NWDSAllocBuf to allocate the result buffer. This buffer does not**

need to be initialized because it is a result buffer.

2. **To search selected attributes, go to step 3. Otherwise, go to step 6.**
3. **Call NWDSAllocBuf to allocate the request buffer.**
4. **Call NWDSInitBuf to initialize the request buffer for a DSV_SEARCH operation.**
5. **Call NWDSPutAttrName once for each attribute name to place the names of the desired attributes into the request buffer.**
6. **To specify a search filter, go to step 7. Otherwise, go to step 11.**
7. **Call NWDSAllocFilter to allocate a filter expression tree.**
8. **Call NWDSAddFilterToken once for each search token to place the search-filter conditions in the expression tree.**
9. **Call NWDSAllocBuf to allocate a filter buffer.**
10. **Call NWDSPutFilter to store the search-filter expression tree in the filter buffer.**
11. **Call NWDSSearch to initiate the search.**
12. **Call NWDSGetObjectCount to determine the number of objects whose information is stored in the buffer.**
13. **Call NWDSGetObjectName to get the name of the current object in the buffer and the count of attributes associated with the object.**
14. **Call NWDSGetAttrName to retrieve the name of the attribute and the count of values associated with the attribute.**
15. **For each value associated with the attribute, call NWDSGetAttrVal to retrieve the value.**
16. **Loop to step 14 until all attribute information for the object has been read.**
17. **Loop to step 13 until the information for all objects in the buffer has been retrieved.**
18. **Call NWDSFreeBuf to free the filter buffer.**
19. **Call NWDSFreeBuf to free the request buffer.**
20. **Call NWDSFreeBuf to free the reply buffer.**

You must pull all information from the result buffer even if you do not plan to use it.

Currently, because of aliasing, searching a subtree can result (1) in duplicate entries or (2) in an infinite loop.

Aborting Iterative Operations

If you need information on aborting an iterative operation, see Controlling Iterations.

Related Topics:

Searching for an NDS Object: Example

NDS Search Introduction

Expression Filters for NDS Searches

NDS Access: Guides

Splitting Partitions

Divide a partition at a specified object by calling **NWDSSplitPartition**.

You can split a partition only if there are no secondary or read-only replicas of the partition. The specified object becomes the root object of the subordinate partition.

Related Topics:

Removing Partitions

NDS Partition: Guides

NDS: Concepts

Access Control Lists

Access to an object and its attributes is based on an Access Control List (ACL). In the NDS Schema, the ACL is an 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.

There are two types of ACLs in NDS. The first one is an ACL; the second is an **Inherited ACL**. The Inherited ACL is an attribute of each partition root object. Standard ACLs are attributes of any object. Both types of ACLs are multivalued attributes that contain access control information.

Each ACL contains a list of subjects for which access has been defined. Each value of an ACL has a subject name, a protected attribute name, and a privilege set:

Subject Name---This field is a complete name of a specific object in NDS, or it can be a special entry name such as "[Inheritance Mask]", "[Public]", "[Root]", "[Creator]", or "[Self]". An ACL entry with "[Inheritance Mask]" in this field is used to mask or filter privileges granted to an object.

Protected Attribute---This field is the name of the attribute that the privilege set applies to. It may instead be an identifier such as "[Entry Rights]", "[All Attributes Rights]" or "[SMS Rights]". If this field is "[Entry Rights]" the access privileges apply to the object that this ACL is an attribute of.

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.

The Inherited ACLs are maintained by a NDS 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 NDS server at the time a new partition is created.

ACLs are created by an administrator to define privilege sets associated with a given object, or a given attribute, or all the attributes of an object, or to define inheritance filters applicable to a given protected object or attribute.

Since ACLs are attributes and are used to define access to individual attributes, they can be used to define access to themselves. The section on managing objects discusses this further.

After creation of a new partition, the administrator may create specific ACL values and define inheritance filters (if so desired) at the object and attribute levels.

Access privileges are given to an object by calling **NWDSModifyObject** to update or add to the value of the ACL attribute.

Related Topics:

Default ACL Templates

Reading the Access Control List: Example

NDS Security: Concepts Guide

Access Privileges

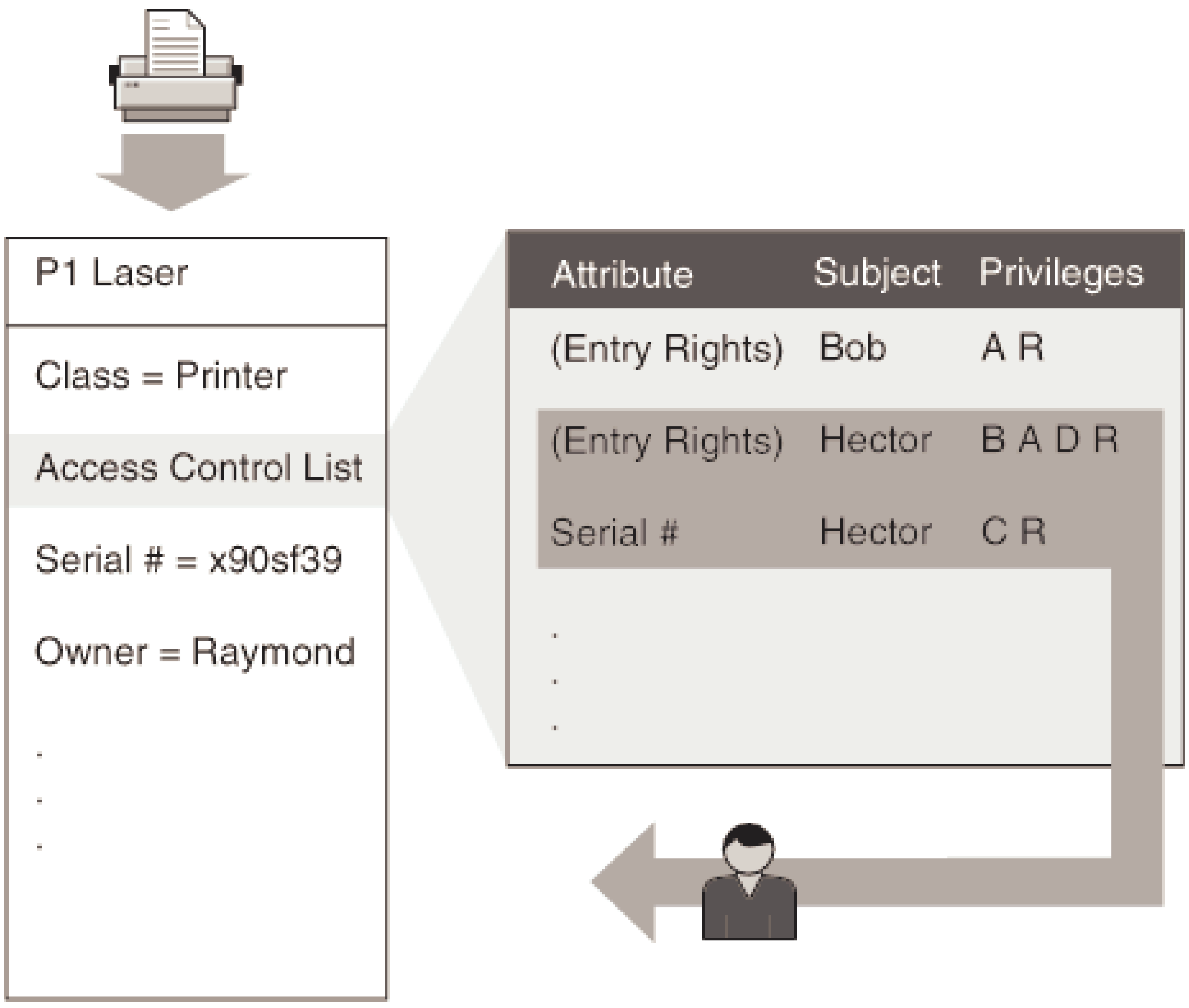
The set of access privileges (or rights) are shown in the following table.

Object Privileges	**Attribute Privileges**	**SMS Privileges**
BROWSE (the object)	COMPARE (the attribute and values)	SCAN
ADD (subordinates)	READ (the attribute and values)	BACKUP
DELETE (the object)	WRITE (add and delete the attribute and values)	RESTORE
RENAME (the object)	SELF (add and delete yourself to the value)	RENAME
SUPERVISOR (implies all entry and all attribute rights)	SUPERVISOR (implies all attribute rights)	DELETE
		ADMINISTRATOR

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.

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 1. Access Control List



The ACL contains the names of subjects and their privileges. Hector appears as subject 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 NDS object**. This access is to the NDS Information Base, not the printer or its queues. The printer and queue access are controlled by the printer's access control mechanisms, not NDS's.

Related Topics:

Rights to the Properties of an NDS Object

NDS Security: Concepts Guide

Addresses Stored in NDS

Like a published directory, NDS 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 NDS. 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.

Related Topics:

Lists Stored in NDS

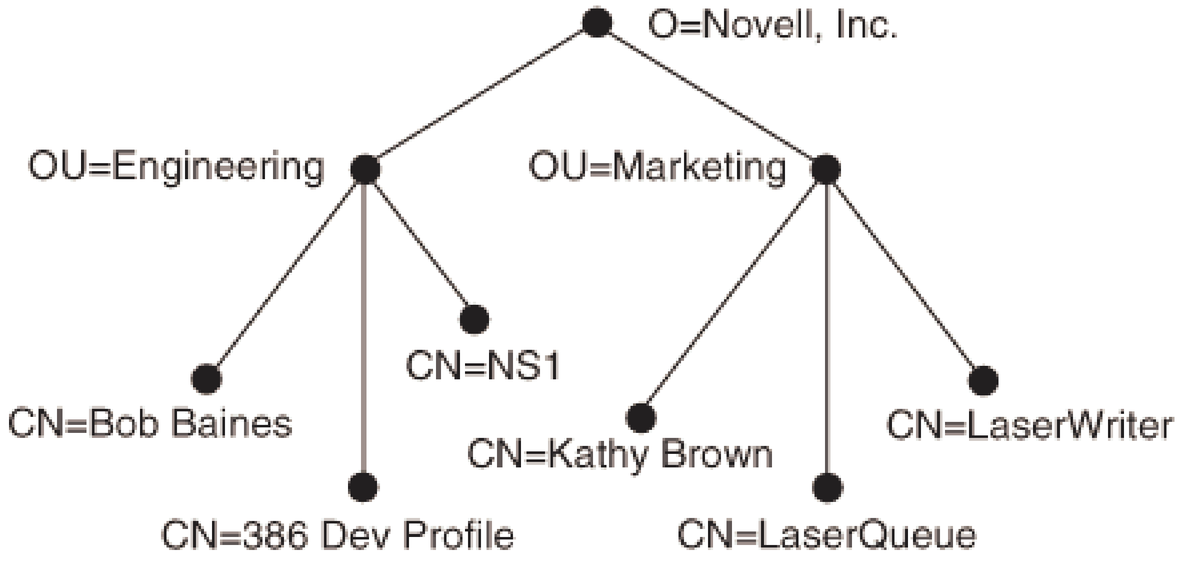
Types of Information Stored in NDS

Administration of Information in NDS

Administration can be understood broadly as managing the information paths within the network. Information paths can be both physical and logical. Attaching a printer to the network is a physical task. Assigning users to a print queue is a logical task. By organizing the names of objects into hierarchies, administrators create logical paths that can facilitate or restrict the flow of information within the network.

The paradigm for the administrative tool is a global tree of objects called the NDS tree, or more commonly referred to simply as NDS. The following figure shows an example tree. The layout of a tree is an administrative responsibility. NDS defines a flexible hierarchy of object classes from which an administrator can choose in developing a tree. This hierarchy definition is called the **NDS Schema**. Objects such as organizations can appear under the root of the tree. Subordinate to these objects can be objects such as organizations or individuals. Consequently, NDS encourages administrators to create a model that reflects the boundaries naturally occurring in their administrative organizations.

Figure 2. A NDS Tree



Information associated with NDS objects, such as addresses and descriptions, can be used to map the NDS tree onto the network's physical topography. For example, a user might be able to query NDS for all the printers associated with the user's department and then read a description printers associated with the user's department and then read a description of each printer and where it is found. A user's ability to view regions within the tree is subject to privileges granted by an administrator.

The NetWare bindery can be used to illustrate the kind of administrative problems addressed by NDS. Although the bindery is an effective tool for managing individual servers, it has significant shortcomings in supporting multiserver networks. Since each bindery is an isolated database, each server maintains its own logical reference for each object it administers. Consequently, on a multiserver network there is no single global reference to refer to a physical object. If a network has 10 servers, one user might be managed as 10 objects.

An additional problem is that the bindery is a flat database that poorly represents the relationships between objects. The bindery can show that a user is a member of a group, but not that the group is part of a department in an organization of a company. Furthermore, some objects, such as printers, are not represented in the bindery at all. All these shortcomings give an administrator a limited and distorted view of a network's organization.

For example, to print a document a user might want to know where the printers in a certain building are. Next, the user needs to know which queues are servicing the desired printer. The user might need to find someone who can grant the user access to the queue. The user might even need to find a supervisor to create a new user account on the server where the queue resides.

In a bindery-based network, the answers to these problems are not easily found. Querying the network can provide some clues, but other clues are likely to come from querying various people. Using NDS, however, an administrator can make all this information conveniently available.

Related Topics:

NDS Compliance to X.500 Standard

Introduction to NDS Development: Guide

Advantages of Loose Consistency

Loose consistency has the advantage of allowing NDS servers to be connected to the network with different types of media. For example, a company might connect parts of its network together by using a satellite link. Data travelling 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, these transmission delays do not interfere with the normal operation of the network because the database is loosely consistent. The new information arrives over the satellite link and is propagated through the network at the next synchronization interval.

Another advantage to loose consistency becomes apparent when communication problems cause a number of the servers on a network to become unavailable. Any changes made to NDS during the time that the servers were out of operation are not lost. When the problem is resolved, the replicas on the affected servers receive updates.

Related Topics:

Developing in a Loosely Consistent Environment

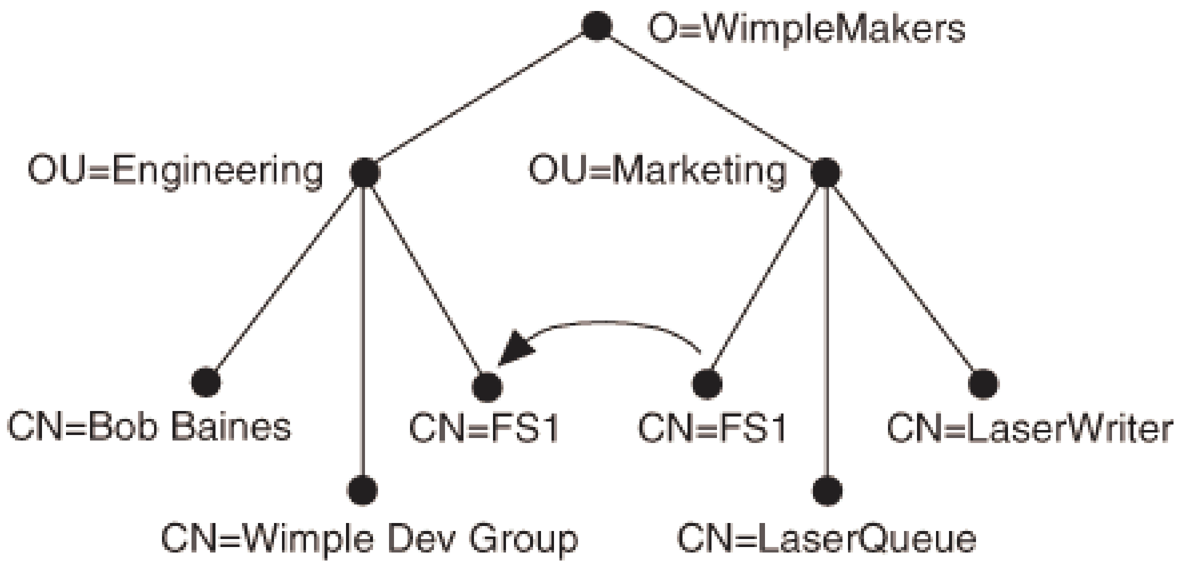
Disappearing NDS Objects

Alias Naming

An alias is a name that can be used as another name for identifying an object. The alias provides an alternate naming path for locating the object in NDS. Any object in NDS 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 NDS 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 3. Using Alias Names



Aliasing is a convenience that has many applications. For example, aliases can be used to simplify searches of NDS. Suppose an administrator creates aliases in a particular subtree for all the modems on the network. Users, then, only have to search this one subtree to receive information about all available modems.

Related Topics:

Name Functions

NDS Context: Guides

All Up To Attribute

This attribute contains a vector of timestamps indicating the time of the last update to the local partition that all other replicas in the partition are guaranteed to have received. The time stamp is incremented with each fully completed update. All entries that are marked to be purged are purged at this time.

Related Topics:

Partition Information

Splitting and Joining Partitions

NDS Partition: Guides

Another Cause of Disappearing NDS Objects

Your program module might fail to find objects that you believe to exist even though your module did not create of the object. If while attempting to read an object you get a NO_SUCH_ENTRY (-601) error when there is good reason to believe the object exists, it might not have been propagated to the replica from which you are reading.

The simplest solution to this problem would be to wait until after the next synchronization interval. If you don't want to wait, you could call **NWDSSyncPartition**, which signals the synchronization engine to update the specified partition. In other words, **NWDSSyncPartition** causes a specified partition to be synchronized without waiting the next synchronization time. This function has four input parameters: the context handle, the name of the server where the partition resides, the name of the partition root, and the number of seconds to wait before beginning the update.

Before calling **NWDSSyncPartition**, you should call **NWDSGetPartitionRoot** to get the name of the partition root. It requires a context handle and the name of the object in the partition, and it returns the name of the partition root. The following code segment shows how to make these calls:

```
ccode = NWDSGetPartitionRoot(context, objectName, partitionRoot);
ccode = NWDSSyncPartition(context, serverName, partitionRoot, 0);
if (ccode)
   printf("Error, sync partition failed. %d\n", ccode);
```

There is one other possible solution to this problem, which might work well in some situations. The partition root object, as a member of the Partition class, has a **Replica** attribute. The Replica attribute is a multivalued attribute that contains a list of servers that store a replica of the partition. After reading the Replica attribute of the partition Root object, you could attempt to read the object you are looking for on each of the servers that contain a replica of that partition until you find the object.

Unfortunately, this last solution does not scale well. On large networks with many replicas of a single partition, this process could take more time than it would save.

Related Topics:

Developing in a Loosely Consistent Environment

Disappearing NDS Objects

Application Programming Interfaces for NDS

The application libraries include many functions, including NDS functions, that can be linked into executable programs. The function requests are completed by using NetWare's DOS Redirector. Some NDS requests can be handled at the workstation and some are sent to an NDS server.

Related Topics:

NDS Workstation Components

Shell Components for NDS

Attribute Constraints

Attribute **constraints** are flags that you can define for an attribute that give the attribute certain characteristics. Constraints can also be described as restrictions affecting the attribute value. Here are some examples of attribute constraints:

DS_SINGLE_VALUED_ATTR

When this flag is set, the attribute can contain only one value. When the flag is cleared, the attribute can contain multiple values.

DS_SYNC_IMMEDIATE

When NDS makes a modification to an attribute with this constraint, other replicas containing the object are synchronized immediately rather than at the next synchronization interval.

DS_PUBLIC_READ

When this flag is set, anyone can read the attribute as long as they are attached to a server, even if they are not authenticated.

The complete list of attribute constraints are defined in NWDSDEFS.H.

Related Topics:

Mandatory and Optional Attributes

NDS Schema: Guides

Attribute Syntax

An attribute syntax defines a data type for the attribute. The standard attribute syntax definitions are contained in NDS Attribute Syntax Definitions. NDS does not allow extension of these syntaxes. You most commonly use syntaxes when you are performing a search. They provide the criteria for the search. The following three examples are selected from the many attribute syntaxes provided by NDS:

SYN_CI_STRING

The Case Ignore String syntax is used in attributes whose values are strings and the case (upper or lower) is not significant.

SYN_INTEGER

The Integer syntax is used for attributes whose values are signed integers.

SYN_TEL_NUMBER

The Telephone Number syntax is used for attributes whose values are telephone numbers.

NDS defines a standard group of attribute syntaxes. An attribute syntax consists of one or more data types for which syntax matching rules have been specified. Matching rules indicate the characteristics that are significant when comparing two values of the same syntax. There are three matching rules: equality, substrings, and ordering. Any number of the rules can apply to the syntax.

To match for **equality**, two equal values must conform to the data type of the attribute syntax. Most syntaxes specify a match for equality. NDS confirms that the values being matched conform to the data type of the syntax. NDS does not attempt to match two values of a syntax that does not specify a match for equality.

To match for **ordering**, a syntax must be open to comparisons of "less than", "equal to", and "greater than."

An example of an attribute syntax is Case Ignore String. Two substrings of this syntax can be matched for equality. When compared, only the characters themselves are tested; the case is ignored.

Search patterns can include asterisks (*).

String syntaxes can include the asterisk (*) wildcard in the pattern to be matched. For example, **N*V*L** would match **NAVAL**, **NAVEL**, or **NOVEL**.

Related Topics:

Attribute Constraints

NDS Attribute Syntax Definitions

NDS Schema: Guides

Attribute Type Abbreviations

For convenience, NDS uses abbreviations for the name types that are used most often. The following table shows the accepted abbreviations for directory attribute types.

Attribute Type	**Abbreviation**
Country	C
Organization	O
Organizational Unit	OU
State or Province Name	S
Locality	L
Common Name	CN
Street Address	SA

NOTE: Types are only assigned to partial names that are typeless. Partial names that include types are not modified.

Related Topics:

Character Encoding

Typeless Names

Default Typing Rule

NDS Context: Guides

NDS Context: Guides

Attribute Type Information

Classes are formed from standard attribute types defined by the schema. You can read existing attribute types and create new ones. You can also remove attribute types, but only nonstandard ones and only if the attribute type isn't assigned to a class. You can't remove any of the standard attribute types.

Once an attribute type has been created, it can't be modified. Attribute type information includes the following:

Attribute Syntax ID

Attribute Flags

ANS.1 ID

NWATTR_INFO contains the attribute type information. The following table lists the attribute flags.

Table auto. Attribute Flags

Flag	Comment
DS_SINGLE_VALUED_ATTR	If TRUE, the attribute is single-valued, otherwise the attribute is multi-valued.
DS_SIZED_ATTR	If TRUE, the attribute has length or range limits.
DS_NONREMOVABLE_ATTR	If TRUE, the attribute can't be deleted. (This flag is for system attributes.)
DS_READ_ONLY_ATTR	If TRUE, clients can't add the attribute to an object, but can read its value.
DS_HIDDEN_ATTR	If TRUE, clients can neither read nor write the attribute.
DS_STRING_ATTR	If TRUE, the attribute syntax is a string. Only attributes with this bit set can be naming attributes.
DS_SYNC_IMMEDIATE	If TRUE, changes to the attribute's value should be immediately synchronized across partitions.
DS_PUBLIC_READ	If TRUE, anyone can read the attribute.
DS_SERVER_READ	If TRUE, server class objects can read

	the attribute even though the privilege to read has not been inherited or explicitly granted.

If the DS_SIZED_ATTR flag is set, the attribute's range or length is limited. If the attribute syntax is a string, the lower and upper range of acceptable lengths is assigned to the attribute. If the attribute is an integer, the range of acceptable values is assigned.

Related Topics:

Attributes and Classes

NDS Attribute Syntax Definitions

NDS Schema: Guides

Attribute Types and Attribute Syntax

An attribute type is based on standard Attribute Syntax and Attribute Constraints. The attribute type definition identifies the attribute syntax to be used and specifies constraints that are imposed on the syntax. For example, a constraint might specify whether the attribute is single- or multi-valued, or what range or size limits are placed on the values defined in the syntax.

An example of an attribute type is *CN (Common Name)* which uses the "Case Ignore String" syntax. *CN (Common Name)* constrains this syntax to a range of from 1 to 64 elements.

Related Topics:

Attribute Syntax

Mandatory and Optional Attributes

Attributes and Classes

NDS Attribute Type Definitions

NDS Attribute Syntax Definitions

NDS Schema: Guides

Attribute Value Comparisons

NWDSCompare compares a given value with the values assigned to a specified attribute. For example, you could ask NDS to compare whether the Member attribute of a particular group is equal to the name of some User. If the comparison is TRUE, the user is a member of the group.

NWDSCompare can be a useful alternative to reading an object's attributes since it requires less effort on your part to examine the results of the request. Also, depending on your access control rights, you may be able to perform a comparison when you can't read the information directly.

You initialize *buf* for a DSV_COMPARE operation and then put an attribute name and value into the buffer. If the proposed value is found, *matched* returns TRUE; otherwise *matched* returns FALSE.

Related Topics:

NDS Search Introduction

NDS Access: Guides

Attributes and Classes

When an attribute is first defined, it is not associated with any object class. Remember, even if you successfully create an attribute definition, it really is not useful until you associate it with a class.

In other words, you could create a set of attributes such as Eyes, Nose, and Mouth. By themselves, they aren't useful. When you define an object class such as Face, you tell it what attributes it is going to have. The attributes now take on a meaning and give dimensions to the object class.

Related Topics:

NDS Attribute Functions

NDS Object Class Definitions

NDS Attribute Type Definitions

Reading an Attribute Definition: Example

NDS Schema: Guides

Authentication

Authentication provides verification that requests received are from valid clients. The term **authentication** is most often used to describe the process of attaching, logging in, authenticating, and licensing a connection. The following topics describe NDS authentication:

Authentication Introduction

Authentication Requirements

Authentication Process

Background Authentication

Authentication of Client Applications

Encryption in NDS

Public and Private Key Pairs

Related Topics:

Authentication Introduction

NDS Access: Guides

NDS Access: Guides

Authentication Introduction

Before you can access any information from the network, you must attach to an NDS™ server and log in to the NDS tree. **Attaching** to a server establishes a connection. When you **login** you establish your credentials. You can access only "public" information in NDS if you have not yet authenticated your connection. Your connection must be **authenticated** in order for you to access all of the NDS information for which you have rights. Lastly, your connection must be **licensed** for you to access file system information.

Authentication has two phases:

Login---Gets the private key.

Authentication---Uses the signature to generate a proof that is used to authenticate (establish identity).

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. (Background authentication refers to subsequent authentication to additional services after the initial login operation.) The user indicates the party to which it needs authentication, and the authentication functions do the rest. The user does not need to retype a password.

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.

Related Topics:

Authentication

Authentication Requirements

Authentication of Client Applications

Most client workstations run LOGIN.EXE when they are booted. LOGIN.EXE establishes an authenticated connection to NDS and a licensed connection to the network server you are attached to.

If your application is meant to run on a client workstation, you might either assume a licensed connection has been established, or run LOGIN.EXE to establish a licensed connection. In either case, your burden has been greatly reduced.

The authentication problem that you will most likely encounter is the need to establish a second connection to another server. For example, your program might read a File object and discover that it refers to the file system of a server to which you have not established a licensed connection.

To gain access to another server, your application can open another connection using **NWDSOpenConnToNDSServer** and then authenticate the connection using **NWDSAuthenticate**.

NWDSAuthenticate does not only authenticate the connection, it also licenses the connection so that you can access the file system. The following code segment demonstrates how to attach and authenticate to a server when you are already authenticated to NDS:

```
ccode = NWDSOpenConnToNDSServer(context, serverName, connHandle);
ccode = NWDSAuthenticate(connHandle, 0, NULL);
if (ccode)
   printf("Error while authenticating connection:%d\n",ccode);
```

Related Topics:

Encryption in NDS

Authentication

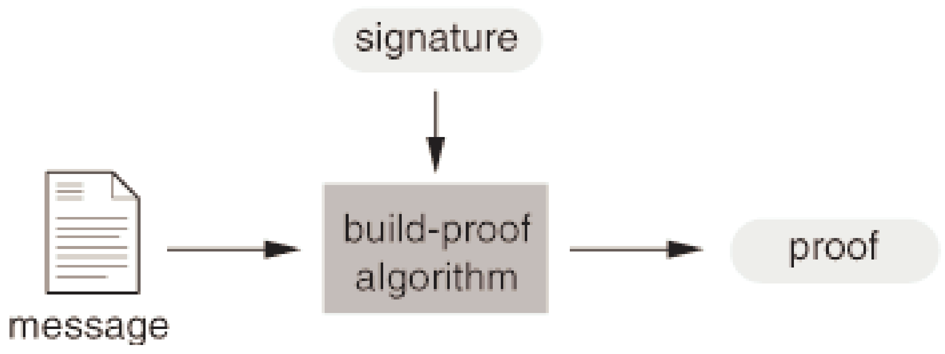
Authenticating to NDS

Authentication Process

Having constructed the credential and the signature, the client agent can now authenticate itself to an NDS server on the network.

The client agent's request for authentication is accompanied by a proof. The proof 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.

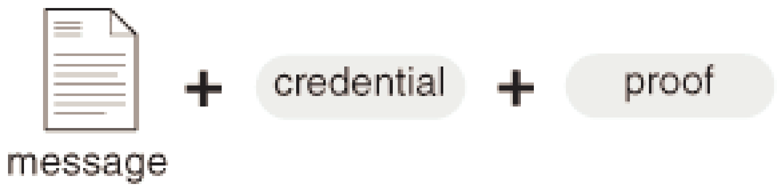
Figure 4. Items Used to Form a Proof


signature
message
build-proof algorithm
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 (a part of NDS) along with the proof and the unencrypted credential. (See the following figure.) When Authentication receives the message, the service is able to mathematically verify that the proof is valid. The proof always assures the recipient that the message has not been modified since it was sent.

Figure 5. 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.

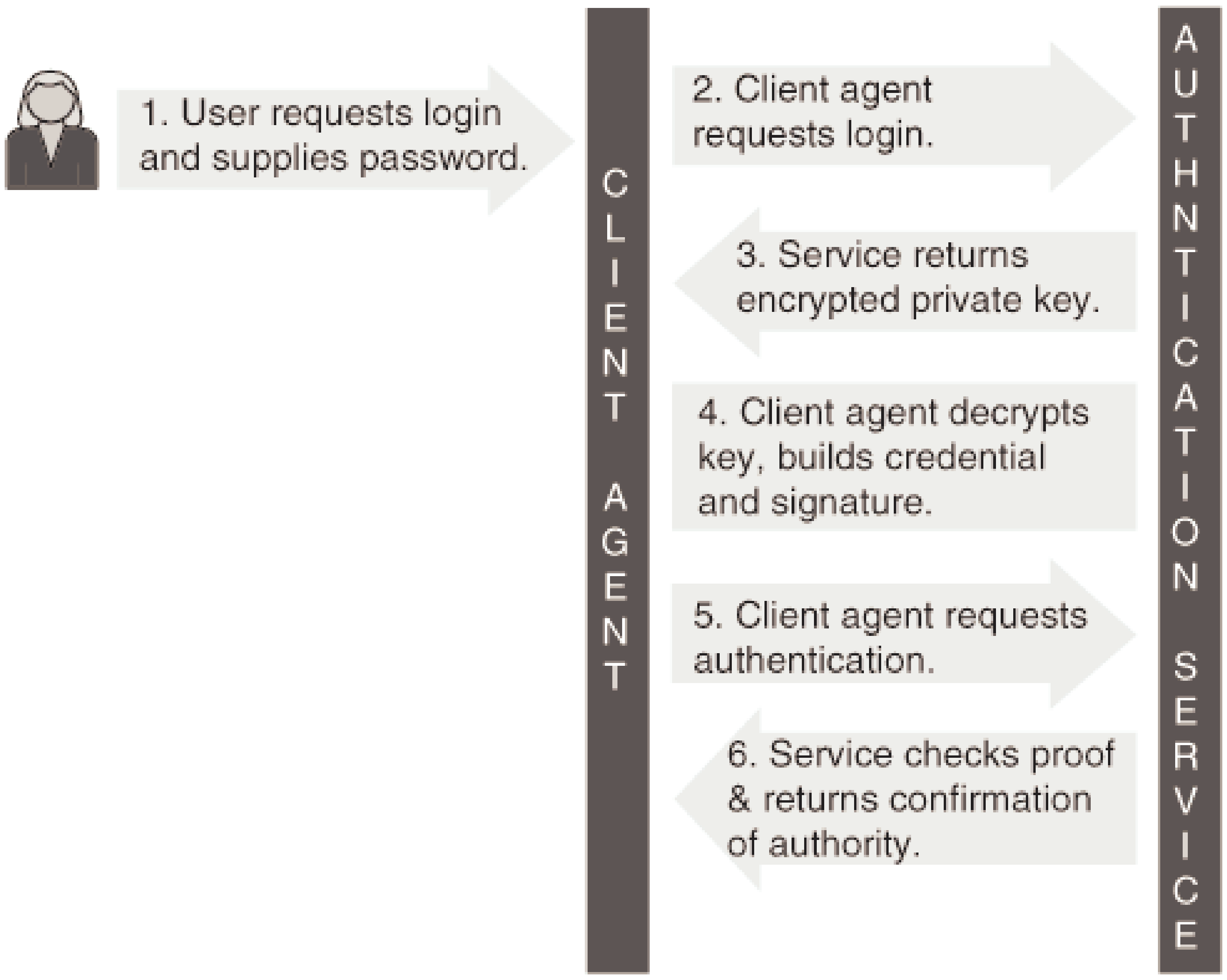
The message pertains to the current session between client and server.

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 6. 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.
AUTHNTICATION SERVICE

Related Topics:

Background Authentication

Authentication

Authentication Requirements

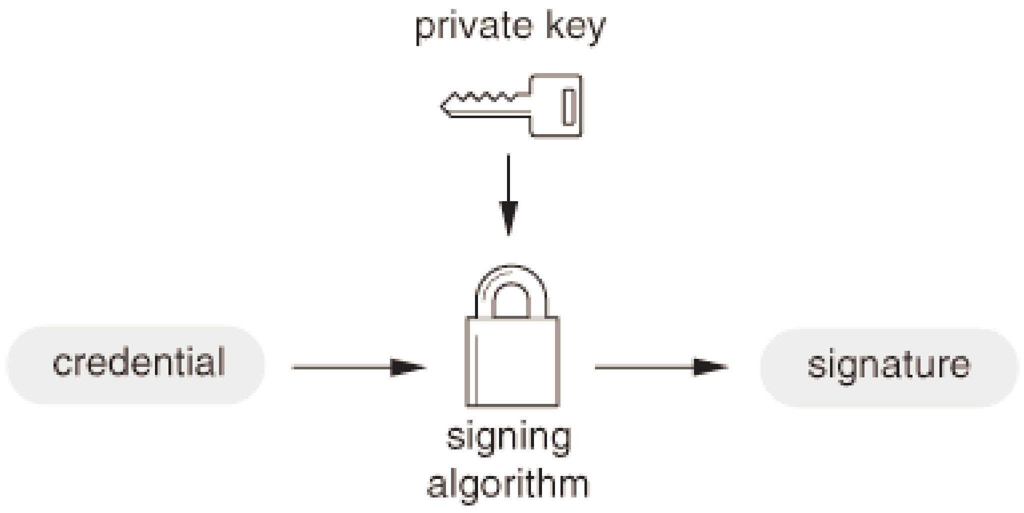
Before you can authenticate to NDS, you need a credential and a signature. You can obtain these by logging in to the network.

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.

This section examines the initial exchange that occurs between Authentication and a client agent to prepare the credential and the signature. In response to a login request, Authentication sends the client agent a private key that has been encrypted and stored in NDS. 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 7. Items Used to Form a Signature



The signature is the important mechanism used in the authentication process. A signature is an encryption of data unique to the client. In many systems, a signature is itself the authentication mechanism and accompanies a message to its destination (with theNetWare® OS, this is not the case as is explained). The 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 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. More is said about proofs below.

Related Topics:

Authentication Process

Authentication

Background Authentication

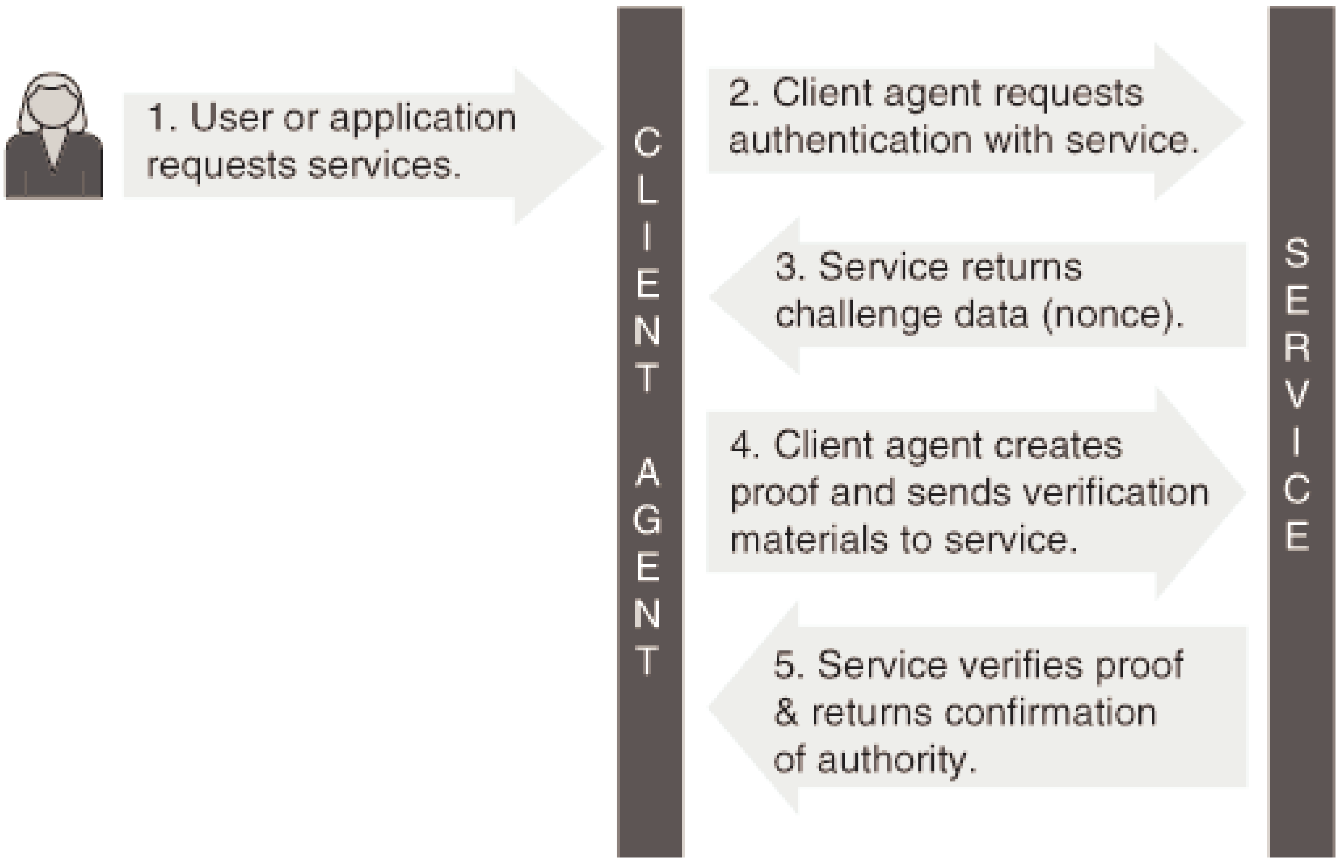
Background authentication with other participating services on the network can be accomplished in a manner similar to the initialization process. In background authentication, the client agent already has the authentication materials (credential and signature) on hand, and wants to authenticate itself 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 (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 8. 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

Related Topics:

Authentication of Client Applications

Authentication

Benefits of NDS

What benefits does NDS provide to users, administrators, and developers?

Some of the key features of NDS are:

Single login/administration

Runtime stability

Ease of administration (single location of data)

Speed

Fault tolerance

With NDS, a user or application need only log in once to access all servers and/or services that reside at the user's security level on the network. NDS manages the authentication details and thus frees applications and users from having to navigate or even understand network configurations. NDS is runtime scalable; whether you want to store four pieces of information or 400, NDS handles it with ease. And NDS provides interfaces that applications and administrators can use to modify NDS information on the fly, without affecting other users.

Administrators find NDS easy to administer because it provides them with a single entry that they can modify to control the network. If an administrator wants to give a new user rights on several systems, he or she can update the NDS data rather than the user data on each system.

NDS provides real fault tolerance. Although data need only be updated in one place, NDS replicates the data for easier access and greater security. Parts or all of the information stored in NDS can be copied to other locations to provide faster service to remote users and redundancy when servers break down.

Thousands of users are becoming networked each day, and these users need directory services in order to easily get access to the network resources they need. Often, their needs will be satisfied by applications---applications of the future. We hope that this *NDS Developer's Guide* will help developers design, create, and launch quality NDS aware applications onto the net.

Related Topics:

Developing in a Loosely Consistent Environment

Introduction to NDS Development: Guide

Browsing NDS

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

Related Topics:

Searching NDS

Retrieval of Information from NDS

Buffer Allocation Types and Related Functions

Value	Allocation Type	Related Function
3	DSV_READ	NWDSExtSyncRead NWDSListAttrsEffectiveRights NWDSRead NWDSReadReferences
4	DSV_COMPARE	NWDSCompare
6	DSV_SEARCH	NWDSExtSyncList NWDSExtSyncSearch NWDSListByClassAndName NWDSListContainers NWDSPutFilter NWDSSearch
7	DSV_ADD_ENTRY	NWDSAddObject
9	DSV_MODIFY_ENTRY	NWDSModifyObject
12	DSV_READ_ATTR_DEF	NWDSReadAttrDef
14	DSV_DEFINE_CLASS	NWDSDefineClass
15	DSV_MODIFY_CLASS_DEF	NWDSModifyClassDef
16	DSV_LIST_CONTAINABLE_CLASSES	NWDSListContainableClasses
18	DSV_ADD_PARTITION	NWDSAddPartition (obsolete 9/97)

Related Topics:

NDS Buffer Management Introduction

Structure of Output Buffers

NDS Buffer Allocation and Initialization Functions

Buffer Size in NDS

To allocate an input or output buffer, call **NWDSAllocBuf**. All input and output buffers are of type Buf_T. You must specify the size of the buffer to allocate. Rather than trying to determine the exact buffer size required for a particular operation, it's usually more convenient to define a standard buffer length that will be adequate for most operations.

> **NOTE:** The size of an output buffer affects how the server processes a request. A server will continue adding data to the buffer until the buffer is full or the request is satisfied. Consequently, if you use a very large input buffer, you may wait longer for initial results than if you use a small buffer. At the same time, a larger buffer may require fewer transmissions than a smaller buffer.

Related Topics:

NDS Buffer Management Introduction

Initialization Operations for NDS Buffers

Preparing NDS Input Buffers

Preparing NDS Output Buffers

Buffers Needed for NDS Searches

You need three types of buffers to search NDS using **NWDSSearch**, two input buffers (called NameBuffer and FilterBuffer in this example) and an output buffer (called ResultBuffer). The purpose of these buffers are summarized below.

NameBuffer

Restricts the search results to certain attributes only.

FilterBuffer

Contains the search expression.

ResultBuffer

Stores the search results.

You need to allocate memory space for all three buffers, and verify that memory is allocated successfully. The following code segment demonstrates how this is done. The data structure for these buffers isBuf_T, which is defined in NWDSBUFT.H.

```
retcode = NWDSAllocBuf(DEFAULT_MESSAGE_LEN,&NameBuffer);
retcode = NWDSAllocBuf(DEFAULT_MESSAGE_LEN,&FilterBuffer);
retcode = NWDSAllocBuf(DEFAULT_MESSAGE_LEN,&ResultBuffer);
```

Since NameBuffer and FilterBuffer are input buffers, they need to be initialized. The output buffer does not need to be initialized.

The following code initializes NameBuffer to the DSV_SEARCH operation.

```
retcode = NWDSInitBuf(context, DSV_SEARCH, NameBuffer);
```

The requested attributes should be stored in NameBuffer. In this case, only the attribute Surname is requested.

```
retcode = NWDSPutAttrName(context, NameBuffer, "Surname");
```

FilterBuffer is initialized to the DSV_SEARCH_FILTER operation:

```
retcode = NWDSInitBuf(context, DSV_SEARCH_FILTER, FilterBuffer);
```

Finally, you should allocate a filter cursor and initialize the cursor to the current insertion point. The data structure for the filter cursor is Filter_Cursor_T, which is defined in NWDSFILT.H.

```
retcode = NWDSAllocFilter(&FilterCursor);
```

Related Topics:

Expression Filters for NDS Searches

Searching NDS

NDS Search Introduction

Canonicalized Names

Canonicalized Names are Context Names that are fully distinguished. Context names are set to canonicalized with DCV_CANONICALIZE_NAMES (see Context Flags).

There two definitions of what a name in **canonical** form means. If you are talking about NDS in general, then a name in canonical form is a name that is fully distinguished and that is fully typed. If you are talking about context flags (specifically the DCV_CANONICALIZE_NAMES flag), a name in canonical form is a name that is fully distinguished only.

The DCV_CANONICALIZE_NAMES flag can appear to be working in a completely opposite way than you might expect. When this flag is set, the libraries expect and return partial names. This means that any name you pass through the functions is going to have the context appended on the end (unless the name is preceded by a period). When this flag is clear, the libraries are going to expect and return fully distinguished names.

If a name in the canonical form means it is fully distinguished, why do the libraries expect and return fully distinguished names when the DCV_CANONICALIZE_NAMES flag is turned off? Think of this flag as a switch that turns on and off a machine that canonicalizes names. When the switch is turned off, the libraries do not touch the name you pass in and just hand it off to NDS. In other words, the name "canonicalizer" is turned off. Because of this, you must pass in fully distinguished names because that is all NDS really knows how to handle. When the switch is turned on, the name "canonicalizer" is turned on and appends the context to any name passed in that is not preceded by a period.

Related Topics:

Default Typing Rule

NDS Context: Guides

Change Types for NWDSPutChange

Table auto. NWDSPutChange Change Types

Value	Change Type	Description
0x00	DS_ADD_ATTRIBUTE	Adds a new attribute to an object. Adding an attribute requires the attribute name. An attempt to add an already existing attribute results in an error.
0x01	DS_REMOVE_ATTRIBUTE	Removes an attribute from an object. Removing an attribute requires to attribute name. An attempt to remove a nonexistent attribute results in an error. This operation is not allowed if the attribute is present in the RDN.
0x02	DS_ADD_VALUE	Adds a value to an attribute. Adding values requires the attribute name. Attribute values inserted in the buffer following the change record are added to the specified attribute. An attempt to add an already existing value results in an error. An attempt to add a value to a nonexistent attribute results in an error.
0x03	DS_REMOVE_VALUE	Removes values from an attribute. Removing values requires the attribute name. Attribute values put in the buffer following this change record are removed from the specified attribute. If the values are not present in the attribute, an error results. This operation is not allowed if one of the values is present in the RDN.
0x04	DS_ADDITIONAL_VALUE	Adds an additional value to a multivalued attribute.
0x05	DS_OVERWRITE_VALUE	Modifies an attribute value without needing to remove the old value first

		and then add the new value.
0x06	DS_CLEAR_ATTRIBUTE	Modifies an attribute value without needing to remove the old value first.
0x07	DS_CLEAR_VALUE	Clears an attribute value without checking to see if the value exists.

Related Topics:

Determining Access Privileges Required for an Operation

NDS Access: Guides

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 computers code page interprets 65 as something entirely different.

To resolve this and other multilingual translation problems, a group of responsible and influential 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 NDS 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.

Related Topics:

Context Settings

NDS Context: Guides

Class Definition Creation

To create a new object class, you must follow a five-step process:

1. **Declare the needed variables.**

2. **Allocate and initialize a request buffer.**
3. **Fill out the class information structure.**
4. **Place the attribute information into the request buffer.**
5. **Create the new object definition.**

Suppose you wanted to create a new class called "Toaster" class. Your first step would be to declare the needed variables:

```
NWDSCCODE      ccode;
NWCLASS_INFO   toastInfo;       // class info structure
NWDS_BUFFER    *toastBuffer;    // request buffer
```

A buffer of type NWDS_BUFFER is needed as a request buffer and a structure of type NWCLASS_INFO is needed to store the class information.

The next step is to allocate and initialize the request buffer:

```
ccode = NWDSAllocBuf(DEFAULT_MESSAGE_LEN, &toastBuffer);
ccode = NWDSInitBuf(context, DSV_DEFINE_CLASS, toastBuffer);
```

Once the buffer has been initialized, your next step is to fill out the class information structure:

```
toastInfo.classFlags = DS_EFFECTIVE_CLASS;   // Set flags to effective
toastInfo.ans1ID.length = 0;                 // Not used in this exampl
toastInfo.ans1ID.data[0] = 0;                // Not used in this exampl
```

With the structure filled out, your next step is to start placing the attribute information into the request buffer. There are five categories of class items that you must place into the class buffer. You call **NWDSBeginClassItem** to start a new category. In fact, this function must be called for each category, even if you are not going to add items to it. To add items to a category, you call **NWDSPutClassItem** for each item added. Here are the five categories of class items in the order they are called:

1. Super class names
2. Containment class names
3. Naming attribute names
4. Mandatory attribute names
5. Optional attribute names

```
ccode = NWDSBeginClassItem(context, toastBuffer);   // Super class name
ccode = NWDSPutClassItem(context, toastBuffer, "Device");

ccode = NWDSBeginClassItem(context, toastBuffer);   // Containment clas
                                        // No new Containment class name
```

```
ccode = NWDSBeginClassItem(context, toastBuffer);   // Naming attribute
                                       // No new Containment class names

ccode = NWDSBeginClassItem(context, toastBuffer);   // Mandatory attrib
                                       // No new Mandatory attributes

ccode = NWDSBeginClassItem(context, toastBuffer);   //Optional attribut
ccode = NWDSPutClassItem(context, toastBuffer, "Toast Setting");
ccode = NWDSPutClassItem(context, toastBuffer, "Status");
```

Mandatory attributes must be added during the creation of a class definition. You cannot go back later and use **NWDSModifyClassDef** to add a mandatory attribute. Nevertheless, you should be very conservative about making an attribute mandatory.

The final step is to create the new object class definition:

```
ccode = NWDSDefineClass(context, "Toaster", &toastInfo, toastBuffer);
if (ccode < 0)
   printf("Error while creating class: %d\n", ccode);
```

NWDSDefineClass requires four arguments: the NDS context handle, the name of the new class, the address of the class information structure, and a pointer to the request buffer.

Related Topics:

Class Maintenance Information

Creating a Class Definition

Creating a Class Definition: Example

Reading a Class Definition: Example

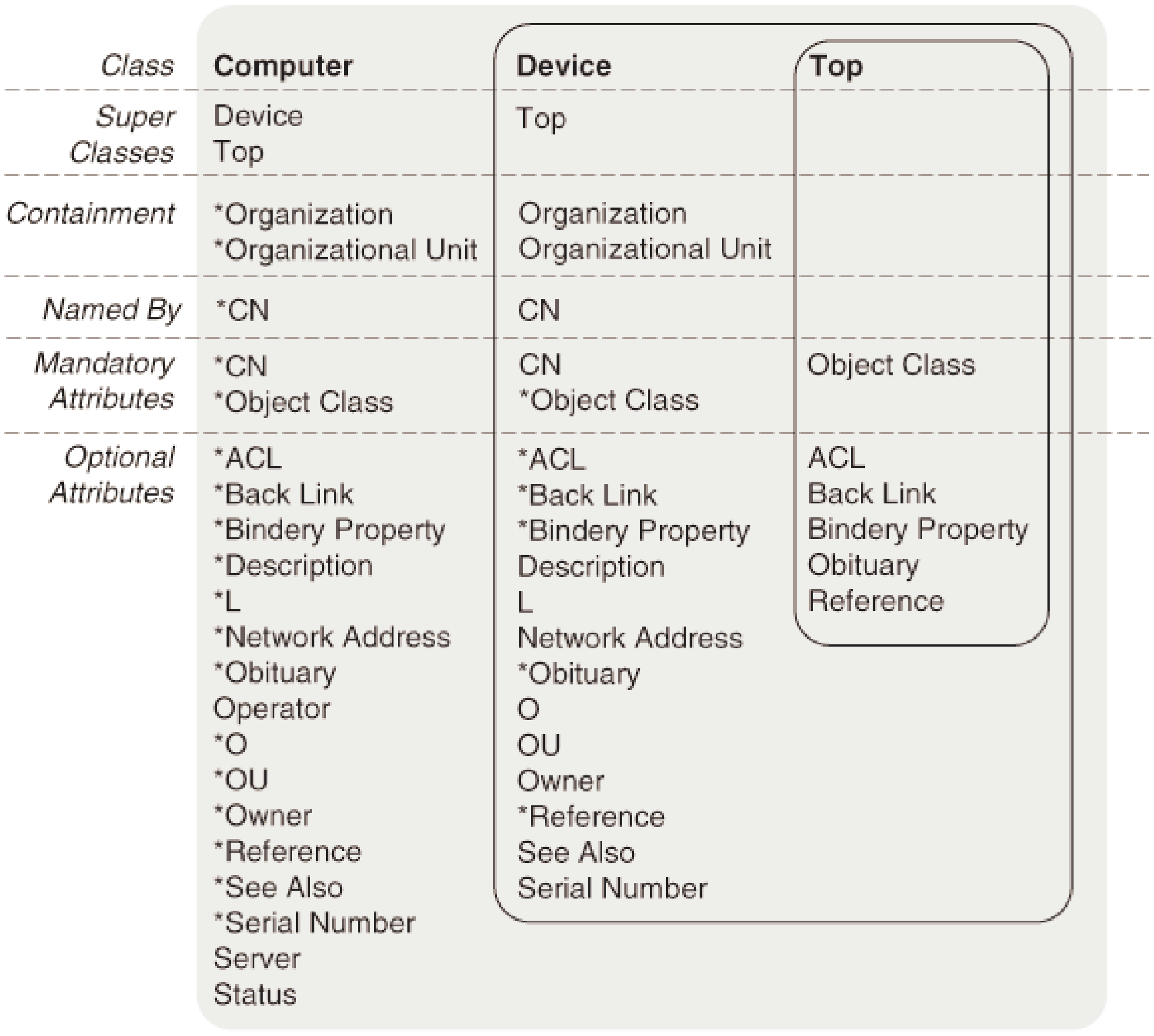
NDS Object Class Definitions

NDS Schema: Guides

Class Inheritance Rules

Class inheritance is determined by a class's super class list. Class inheritance is illustrated in the following figure by the relationship between the classes Top, Device (Class) and Computer. Computer has Device as a super class and inherits all the features defined by Device. Device and Computer inherit from Top, because Top is a superclass of Device.

Figure 9. NetWare 4.0 Class Inheritance for Computer Class



Class	Computer	Device	Top
Super Classes	Device Top	Top	
Containment	*Organization *Organizational Unit	Organization Organizational Unit	
Named By	*CN	CN	
Mandatory Attributes	*CN *Object Class	CN *Object Class	Object Class
Optional Attributes	*ACL *Back Link *Bindery Property *Description *L *Network Address *Obituary Operator *O *OU *Owner *Reference *See Also *Serial Number Server Status	*ACL *Back Link *Bindery Property Description L Network Address *Obituary O OU Owner *Reference See Also Serial Number	ACL Back Link Bindery Property Obituary Reference

* Inherited from Super Classes

A computer is a special kind of device. The Computer class uses all of the attributes from the Device class but adds several attributes to accommodate the needs of computers. A different type of device adds different attributes than a computer, but the new class and the Computer class inherits all of the than a computer, but the new class and the Computer class inherits all of the attributes from the Device class.

In the example, the Computer class inherits all of the attributes defined by the Device class, including the containment classes Organization and Organizational Unit, and a naming attribute, *Common Name*. To the definition of Device, Computer adds a few optional attributes (such as Server and Status) that accommodate the needs of computer objects.

According to the object class definition, a Computer object could appear only as a subordinate of objects belonging to either the Organization or the Organizational Unit class. The object would be recognized by its common name and might be assigned various optional attributes. In this case, Computer refines the definition of the Device to serve a particular class of devices.

Related Topics:

Structure Rules

Super Class Structure

Class Maintenance Information

In addition to the basic types of information used to construct a class, NDS stores additional information to maintain the class. NWCLASS_INFO contains the class maintenance information.

Class maintenance information includes the classes ASN.1 ID and a set of class flags identifying the status of the class. The following table lists the set of class flags.

Table auto. Class Flags

Flag	Comment
DS_CONTAINER_CLASS	If TRUE, objects of the class can have subordinates.
DS_EFFECTIVE_CLASS	If TRUE, the class can be used as a base class.
DS_NONREMOVABLE- _CLASS	If TRUE, the class definition can't be removed from the schema.
DS_AMBIGUOUS_NAMING	If TRUE, the class can't be used as a base class.
DS_AMBIGUOUS_CONTAINMENT	If TRUE, the class can't be used as a base class.

The ASN.1 ID is an object identifier created according to the rules specified in the ASN.1 standard. The object identifier is encoded using the ASN.1-BER rules. It is optional. If no ASN.1 object identifier has been registered for the attribute, a zero-length octet string is specified.

Related Topics:

Restrictions on New Classes

NDS Schema: Guides

Configurable NDS Functions

NWDSRead is a representative of NDS functions that are configurable. (Others include **NWDSSearch**, **NWDSReadClassDef**, and **NWDSReadAttrDef**.) These functions let you configure the results to be returned.

In configurable functions, *infoType* and *allAttrs* act as configuration flags.

infoType lets you define the extent of the information returned. If*infoType* is DS_ATTRIBUTE_NAMES (0), only the names of attributes are returned. If *infoType* is DS_ATTRIBUTE_VALUES (1), the attribute names are accompanied by their attribute values.

allAttrs defines the scope of the request. If *allAttrs* is TRUE (1), information is returned for all attributes associated with the object. If *allAttrs* is FALSE (0), you must specify the attributes to be read.

Results are returned in an output buffer that must be allocated by the application. Depending on the size of the output buffer and the amount of information returned, you may need to call a function several times to retrieve all the results.

Related Topics:

Controlling Iterations

NDS: Functions

NDS Access: Guides

Contained By Classes

The following table shows both types of objects and the object types they can be contained by.

Object Class	**Contained Classes**
AFP Server	Organization Organizational Unit

Alias	Special
Bindery Object	Organization Organizational Unit
Bindery Queue	Organization Organizational Unit
Computer	Organization Organizational Unit
Country	Top
Directory Map	Organization Organizational Unit
External Entity	Organization Organizational Unit
Group	Organization Organizational Unit
List	Organization Organizational Unit
Locality	Country Organization Organizational Unit
Message Routing Group (Class)	Organization Organizational Unit
NCP Server	Organization Organizational Unit
Organization	Country Locality Top
Organizational Role	Organization Organizational Unit
Organizational Unit	Locality Organization Organizational Unit
Print Server (Class)	Organization Organizational Unit
Printer (Class)	Organization Organizational Unit
Profile (Class)	Organization Organizational Unit
Queue (Class)	Organization Organizational Unit
Top	None

Unknown (Class)	None
User (Class)	Organization Organizational Unit
Volume (Class)	Organization Organizational Unit

Noneffective classes are not represented in this chapter since they cannot be used to create objects in NDS. They are only used to define rules for other object classes to inherit.

Related Topics:

Containment Classes

Structure Rules

Containment Classes

Objects that can contain other objects are called **container objects**. Container objects are the branches of the NDS tree and provide a structure that is similar to a directory in a file system. Objects that cannot contain other objects are called **noncontainer** or **leaf objects**. Leaf objects represent the actual network resources that perform some function in the NDS tree, such as Users, Printers, Modems, Servers, or Volumes.

Containment rules of an object limit the possible locations for the object in the NDS tree. An object can be contained by only those objects listed among the containment classes of the object. Container objects are also called **parent objects**.

For each object class, a list of containment classes specifies where an object of that class may appear in the hierarchical structure of the NDS tree. An object can be immediately subordinate to only those objects whose classes appear in the containment list of the object's expanded class definition. Containment classes limit the possible locations of an object in the NDS tree, thus restricting the order and types of partial names that appear in the object's complete name.

Containment helps to ensure that the NDS tree expands in a consistent and logical fashion. For example, a Country object can only be subordinate to the root of the tree. Consequently, where the name of a country is present in a complete name, it is always the most significant component. The name of a country can never be subordinate to the name of an organization or a locality. Other objects, however, can also be subordinate to the root, so a country name is not **necessarily** the most significant component of a complete name.

Containment classes, while helping to control the structure of NDS, must also be flexible enough to accommodate a variety of organizational problems. An example is the relationship between the classes Organization and Locality. Each class specifies the other as a containment class. This allows an administrator to decide which hierarchical order best represents his organization.

The following table shows the containment classes, and the object types that they can contain.

Object Class	**Contained Classes**
Top	Country
Organization	Organizational Unit
Country	Locality
Organization	Organizational Unit
Locality	Organization
Organizationa l Unit	Organization
Organizationa l Unit	Leaf objects
Locality	Organizational Unit

Related Topics:

Name Rules

Contained By Classes

Structure Rules

Context Flags

Context Flags, which are associated with the DCK_FLAGS context key, determine how requests made to NDS are processed and how data is returned from the functions. For more information seeContext Key Values.

Each context you create has its own set of flags that govern the way the functions return information. The data type that defines a context flag is a nuint32 (unsigned 32-bit integer).

There are two flags that you will deal with all the time, DCV_TYPELESS_NAMES (see Typeless Names) and DCV_CANONICALIZE_NAMES (see Canonicalized Names).

The following table defines the flags of a context:

Define Name	**Bit**	**Initia**	**Definition**

		l Valu e	
DCV_DEREF_ALIASES	0	1	If set, libraries act on objects referenced by alias, not on the alias object.
DCV_XLATE_STRINGS	1	1	If set, libraries return values in the local code page.
DCV_TYPELESS_NAMES	2	0	If set, libraries return typeless names.
DCV_ASYNC_MODE	3	n/a	Reserved.
DCV_CANONICALIZE_NAMES	4	1	If set, libraries canonicalize names passed in and return partial names.
DCV_DEREF_BASE_CLASS	5	0	If set, an alias object's base class is set with the dereferenced value instead of "Alias". This flag affects List, Read, and Search operations.
DCV_DISALLOW_REFERRALS	6	0	Is set, the NDS agent is disallowed from referring the client agent to a differenct NDS agent. This restricts the name resolution to the NDS agent being addressed.

The following example demonstrates how to work with context flags. First, we create a context. We then read the flags, clear the DCV_DEREF_ALIASES flag, set the DCV_TYPELESS_NAMES flag. When finished, we free the context.

```
NWDSCCODE            ccode;
NWDSContextHandle    context;
nuint32              flags;

context = NWDSCreateContext();
if(context == (NWDSContextHandle)ERR_CONTEXT_CREATION)
    /* HANDLE ERROR */

ccode = NWDSGetContext(context, DCK_FLAGS, &flags);
if(ccode)
    /* HANDLE ERROR */
```

```
flags &= ~DCV_DEREF_ALIASES;
flags |= DCV_TYPELESS_NAMES;

ccode = NWDSSetContext(context, DCK_FLAGS, &flags);
if(ccode)
    /* HANDLE ERROR */

/* NDS Code */

NWDSFreeContext(context);
```

Related Topics:

Context Name

NDS Context: Guides

Context Key Values

Context Key Values explains the key parameter required for Reading Context Information or for Modifying Context Information.

The two functions for getting and setting different aspects of the context are **NWDSGetContext** and **NWDSSetContext**. Because both of these functions require a *key* value as a parameter.

The *key* parameter acts as an index into the context structure. The keys that can be used here are defined in the following table along with the description of what each key does. The two parts of the context that you deal with the most are the context flags and the context name.

Table auto. Context Key Table

Key	Name	Description
DCK_FLAGS	Context Flags	Determines how requests made to NDS are processed and how data is returned from the functions.
DCK_CONFIDENCE	Confide nce	Determines what replica type to use when processing requests.
DCK_NAME_CONTE XT	Context Name	The current view of the NDS tree. Much as your current path changes as you run applications, your context changes as you access data in the tree.
DCK_TRANSPORT_T YPE	Transpor t Type	Allows for an implementation of NDS on some other protocol to interface with NDS.
DCK_REFERRAL_SC OPE	Referral Scope	For future use. NDS does not currently use this variable.

OPE	Scope	currently use this variable.
DCK_LAST_CONNEC TION	Last Connecti on	Contains the connection handle of the last server to which the library sent a request. This variable is cleared when the tree name is changed.
DCK_TREE_NAME	Tree Name	Contains the name of the tree in the current context.

Related Topics:

Context Flags

NDS Context: Guides

Context Name

Context Name is one of the Context Key Values. Changing the Context Name changes your location in the NDS directory tree.

The context name is the most visible part of the context structure. The context name is the parameter that is set in the client configuration. Changing this value is like changing directories on a file system.

The context name is stored in memory as a text string, so any value is valid. The context could be set to "Mickey Mouse and Goofy Too" and **NWDSSetContext** would not fail. If you set the context without types, the context is returned without types.

The following code segment shows how to get and set the context name.

```
NWDSCCODE            ccode;
NWDSContextHandle    context;
nstr8                context_name[MAX_DN_CHARS+1];

context = NWDSCreateContext();
if(context == (NWDSContextHandle)ERR_CONTEXT_CREATION)
    /* HANDLE ERROR */

ccode = NWDSGetContext(context, DCK_NAME_CONTEXT, context_name);
if(ccode)
    /* HANDLE ERROR */

strcpy(context_name, "OU=ENGINEERING.O=ACME");

ccode = NWDSSetContext(context, DCK_NAME_CONTEXT, context_name);
if(ccode)
    /* HANDLE ERROR */

/* NDS Code */
```

```
NWDSFreeContext(context);
```

Related Topics:

Country Name

NDS Context: Guides

Context Settings

Context Settings briefly lists the possible Context Key Values and Context Flags with their default values if applicable.

When you create a new context variable with **NWDSCreateContextHandle**, you get a handle to the newly created context. The context handle settings are as follows:

DCK_FLAGS = The context flags

- DCV_DEREF_ALIASES = ON
- DCV_XLATE_STRINGS = ON
- DCV_TYPELESS_NAMES = OFF
- DCV_ASYNC_MODE = OFF
- DCV_CANONICALIZE_NAMES = ON
- DCV_DEREF_BASE_CLASS = OFF
- DCV_DISALLOW_REFERRALS = OFF

DCK_CONFIDENCE = DCV_LOW_CONF (0)

DCK_NAME_CONTEXT = The default name context held by the requester or, if the platform is NLM, the bindery context of the server. Otherwise, set to [Root].

DCK_TRANSPORT_TYPE = NT_IPX (0)

DCK_REFERRAL_SCOPE = DCV_ANY_SCOPE (0)

DCK_DEREF_BASE_CLASS = -1 (Invalid connection)

DCK_TREE_NAME = The preferred tree according to the requester or, if the platform is NLM, the name of the tree to which the server belongs.

Related Topics:

NDS Context

NDS Context: Guides

Controlling Iterations

The *iterationHandle* parameter controls the retrieval of output data that is larger than the output buffer pointed to by the *subordinates* parameter. Before the initial call to an iterative function like **NWDSList**, you should set the contents of the iteration handle pointed to by the *iterationHandle* parameter to NO_MORE_ITERATIONS.

When the function returns from its initial call, if the output buffer holds the complete results, the value of *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, make another call to the function to obtain another portion of the results. When the results are completely retrieved, the value of the iteration handle is once more set to NO_MORE_ITERATIONS.

If you want to end an iterative operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_LIST to free memory associated with the list operation.

Related Topics:

Read Requests

NDS Access: Guides

Convergence Attribute

With multiple replicas of the same information existing on the network, it is inevitable that for brief periods of time the information in one replica differs from that in the others. When the information in a partition is changed, that replica is not synchronized with the other replicas. When this occurs, an immediate (within 10 seconds) effort is initiated to update all replicas so they all have the same information. Any writable replica can be changed and initiate this synchronization process.

Related Topics:

All Up To Attribute

Partition Information

Country Name

The client agent expands any partial names into complete names before submitting the names to NDS. If part of the complete name is a Country name, the client agent also checks the Country name length. If the Country name length is not exactly two letters, the client agent does not submit the request to NDS. This is because all valid Country names are two letters in length.

The client agent does not check the value of the abbreviation to determine if it is one of the valid Country names. The client agent considers any two letters to be a legal Country name (although the letters themselves may not specify a legal Country name).

The two-letter country codes are taken from ISO 3166.

Related Topics:

Attribute Type Abbreviations

NDS Context: Guides

Default ACL Templates

In order to guarantee a minimum amount of functionality and access security for newly created objects NDS provides a few default ACL templates. These templates are included in the Schema definitions. If during creation of an object no ACLs are specified then the default ACL templates are used to create an ACL for the object. The templates specify ACL values that will give the object being created a functional degree of NDS security.

If an ACL attribute is specifically created when an object is created then the templates are not used at all. NDS automatically checks to see if an ACL is created and creates one based on the templates if it does not exist.

Related Topics:

Inheritance in NDS

Access Control Lists

NDS Security: Concepts Guide

Default Typing Rule

If an application requests the library to apply name types to a typeless name, the following rule is applied:

Default Typing Rule

The most significant (rightmost) component is an Organization (O).

The least significant (leftmost) component is a Common Name (CN).

All intervening components (in the middle) are Organizational Units (OU).

The library adds types to any components of a name that are passed into the API without a type. If you pass in a name like "JRoss.Engineering.ACME" the libraries prepend types to each component and forward the name to NDS as "CN=JRoss.OU=Engineering.O=ACME". The library does not know the correct types to apply to names nor can it "look them up"; it simply follows the Default Typing Rule.

> **NOTE:** Fully typed named were required for NetWare 4.0 and 4.01 only. Support for typeless names is available in all other versions of NetWare 4.x.

Related Topics:

Name Expansion

NDS Context: Guides

DEFAULT_MESSAGE_LEN Constant

The DEFAULT_MESSAGE_LEN constant is defined as 4 K. You could allocate more or less than this, but there is a trade-off. The size of an output buffer affects how the server processes a request. A server continues adding data to the buffer until either the buffer is full or the request is satisfied. Consequently, if you use a very large input buffer, you might wait longer for initial results than if you use a small buffer. However, a larger buffer might require fewer transmissions than a smaller buffer.

Related Topics:

Buffer Allocation Types and Related Functions

Preparing NDS Input Buffers

Preparing NDS Output Buffers

NDS Buffer Management Introduction

Descriptions Stored in NDS

NDS can be used to store object descriptions. A **Description** property can be used to associate a descriptive string with an NDS 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.

Related Topics:

Retrieval of Information from NDS

Types of Information Stored in NDS

Determining Access Privileges Required for an Operation

Since a subject can receive 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. Each of the following cases explains the privileges required at both the object and attribute level to perform a particular operation. The following table summarizes the discussion on how privileges are applied.

Table auto. 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
Delete object	Delete	AND	Write on each present attribute
Move object	Delete (at the source location); Add (at the	AND	Write on each present attribute

	destination)		
Write self	NONE	AN D	Self
Modify Name (RDN)	Rename	AN D	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, the Supervisor privilege if inherited can be restricted by an inheritance mask.

Related Topics:

NDS List Functions

NDS Access: Guides

Developing in a Loosely Consistent Environment

NDS™ is described as a loosely consistent environment, which means that there is no guarantee that all replicas hold the same data at any one moment in time. In other words, partition replicas are not instantaneously updated, and a change made to one replica must be synchronized with other replicas.

The synchronization interval is established by the network administrator and can be as short as ten seconds or as long as five minutes. In computer program terms, even an interval of ten seconds seems like an eternity.

For a developer who is new to the NDS environment, developing can present many new challenges. A program that works well in the test environment can suddenly become unreliable when installed on a real network. Or a program that seems to work on a small network might not work at all on a large busy network. Taking time now to consider some of the implications of working in a loosely consistent environment can save many hours of grief later.

Related Topics:

Advantages of Loose Consistency

Disappearing NDS Objects

Another Cause of Disappearing NDS Objects

Introduction to NDS Development: Guide

Directory Information Base

The Directory Information Base (DIB) is another name for NDS. The DIB is implemented as a set of data files managed by the NDS record manager.

Related Topics:

NDS Architecture Introduction

NDS Architecture: Concepts Guide

Disappearing NDS Objects

Let's suppose that your program creates an object in the NDS tree. After creating the object, the program collects data that it needs to modify the attributes of the object. This data could come from any server on the network. Now that your program has the data it needs, the program calls **NWDSModifyObject**, which returns a NO_SUCH_ENTRY (-601) error.

"You mean to tell me that even if I successfully create an object, I might not be able to find it when I want to access it?" Yes. In the example cited above, the operations used to correct the data for the attributes of the new object changed the context variable that pointed to the server that contained the replica where the new object was created. When the program attempted to modify the object, it accessed another server containing a copy of the right replica, but this replica had not yet been updated with the information for the new object.

You can avoid this problem by caching the contents of the context variable called DCK_LAST_CONNECTION immediately after creating the object. This variable contains the name of the server where you created the object.

Another way to avoid this problem would be to create a separate context handle for operations that deal with other objects. This would insure the context handle for the object that was just added would not be changed.

Related Topics:

Another Cause of Disappearing NDS Objects

Developing in a Loosely Consistent Environment

Distribution of Access Control Information

By inheritance, access control information defined at one point in the NDS 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 Inherited Rights Filter.)

The partitioning of NDS 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. NDS alleviates this problem through Inherited Access Control Lists (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.

Related Topics:

Distribution of Schema Information

NDS Partition: Guides

Distribution of Schema Information

Schema information represents an important category of partition information. The schema defines the data types for information that can be added to the NDS tree.

Each name server maintains its own copy of the NDS schema. Changes to the NDS schema can be made **only** on a server that stores a writable replica of the NDS directory root ([root]) partition.

The background process on the server where the changes are made propagates the schema update to all servers in the tree.

Related Topics:

Distribution of Access Control Information

NDS Partition: Guides

Distribution of the NDS Schema

The NDS schema is global. Each server stores a replica of the schema in its entirety. The schema replica is stored separately from the partitions that contain directory objects. Changes to any one schema replica are propagated to the other replicas. You can perform modifications to the schema only through a server that stores a writable replica of the root partition. Servers storing read-only replicas of the root partition can read but not modify schema information.

Related Topics:

NDS Schema Extensions

NDS Schema: Guides

DSI Flags

The DSI Flags affect the information returned by the **NWDSSetContext**, **NWDSGetContext**, **NWDSReadObjectInfo**, **NWDSList**, **NWDSReadObjectDSIInfo**, and **NWDSGetDSIInfo** functions.

If you remove the default flags from the context handle, the **NWDSReadObjectInfo** and **NWDSGetDSIInfo** functions that follow the **NWDSList** function will be affected.

If you add flags to the context handle, you can access the additional information by calling the **NWDSReadObjectDSIInfo** function or by calling the **NWDSList**, **NWDSGetObjectNameAndInfo**, and **NWDSGetDSIInfo** functions in this specific order.

Valid DSI Flags follow:

00001 h	uint32	DSI_OUTPUT_FIELDS
00002 h	uint32	DSI_ENTRY_ID
00004 h	uint32	DSI_ENTRY_FLAGS: `0001h DS_ALIAS_ENTRY 0002h DS_PARTITION_ROOT 0004h DS_CONTAINER_ENTRY 0008h DS_CONTAINER_ALIAS 0010h DS_MATCHES_LIST_FILTER 0020h DS_REFERENCE_ENTRY 0040h DS_40X_REFERENCE_ENTRY`
00008 h	uint32	DSI_SUBORDINATE_COUNT: 1 if unknown
00010 h	Time	DSI_MODIFICATION_TIME: 0 if unknown
00020 h	Timestamp	DSI_MODIFICATION_TIMESTAMP: 0 if unknown
00040 h	Timestamp	DSI_CREATION_TIMESTAMP: 0 if unknown
00080 h	uint32	DSI_PARTITION_ROOT_ID: Entry ID of the partition root entry
00100 h	uint32	DSI_PARENT_ID

00200h	uint32	DSI_REVISION_COUNT
00400h	uint32	DSI_REPLICA_TYPE: `0 RT_MASTER 1 RT_SECONDARY 2 RT_READONLY 3 RT_SUBREF`
00800h	Ustring	DSI_BASE_CLASS
01000h	Ustring	DSI_ENTRY_RDN
02000h	Ustring	DSI_ENTRY_DN
04000h	Ustring	DSI_PARTITION_ROOT_DN
08000h	Ustring	DSI_PARENT_DN
10000h	Time	DSI_PURGE_TIME
20000h	Ustring	DSI_DEREFERENCE_BASE_CLASS

Related Topics:

DSP Flags

DSP Flags

The DSP Flags affect the information returned by the **NWDSListPartitionsExtInfo**, **NWDSGetPartitionExtInfoPtr**, and **NWDSGetPartitionExtInfo** functions and do not affect any previous functions.

You can access the additional information by calling the **NWDSListPartitionsExtInfo**, **NWDSGetPartitionExtInfoPtr**, and **NWDSGetPartitionExtInfo** functions in this specific order.

Related Topics:

DSI Flags

Effective and Noneffective Classes

Object classes can be either effective or noneffective. The term **effective class** means that you can actually create an instance of the defined object in the NDS tree. The term **noneffective class** means that the class is only used to define other classes. You cannot create an object of a noneffective class.

The Computer class, for example, is an effective class. You could create a Computer object on the NDS tree using the Computer class. The Device class is a noneffective class. You could not create a Device object because it would have no real function. However, the Device class is a super class of the Computer class and helps to define the attributes needed by the Computer class.

Effective or noneffective status is assigned to a class when the class is defined by **NWDSDefineClass**. Schema Services define a structure, NWCLASS_INFO, that is used to flag the class effective or noneffective. This value cannot be modified after the class is created.

Related Topics:

Class Definition Creation

Creating a Class Definition: Example

Reading a Class Definition: Example

Super Class Structure

Effective Rights Calculations

Everything described in the sections titled Rights to an NDS Object, Rights to the Properties of an NDS Object, and Giving Rights in NDS is used by NDS to calculate the Effective Rights of an object to another object. Inherited Rights Filters can mask out rights that are inherited from higher up in the tree, but may be overridden by explicitly assigned rights and security equivalences below the Inherited Rights Filter. The important thing is to be familiar with how each of the rights-assignment methods work. Then you will understand how to design an NDS 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.

Related Topics:

Trustees

Reading Effective Rights: Example

NDS Security: Concepts Guide

Encryption in NDS

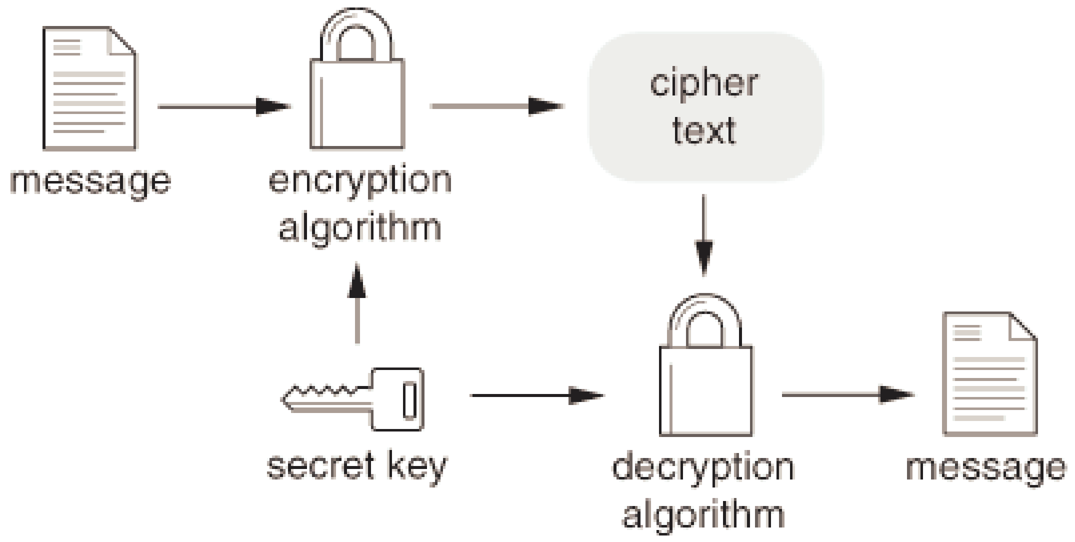
Authentication relies on **encryption** systems, procedures that allow information to be transmitted in unreadable forms. (The information is unreadable in the sense that, in its encrypted form, it is not meaningful to anyone except to someone who has the key to decrypt it.) Authentication uses encryption 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 is called **decryption**. Decryption reverses the encryption process. 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.

Encryption systems can be symmetrical or asymmetrical. In a symmetrical system, the same key is used in both the encryption and decryption processes (The following figure shows the symmetrical encryption process). Symmetrical encryption is fast and efficient. However, it has a significant drawback: the key itself must somehow be distributed among the parties exchanging messages. As a result, a medium other than the network, such as a courier, is required to safely distribute the key among the participants. Arranging a special distribution process can be both inconvenient and risky.

Figure 10. Symmetrical Encryption



An asymmetrical system solves this problem by using 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. Hence, asymmetrical encryption is referred to as **public key encryption**.

The one that wants to receive messages generates a key pair and distributes the public key. Those that want to send encrypted messages do so with the public key. Only the holder of the private key can decrypt the messages.

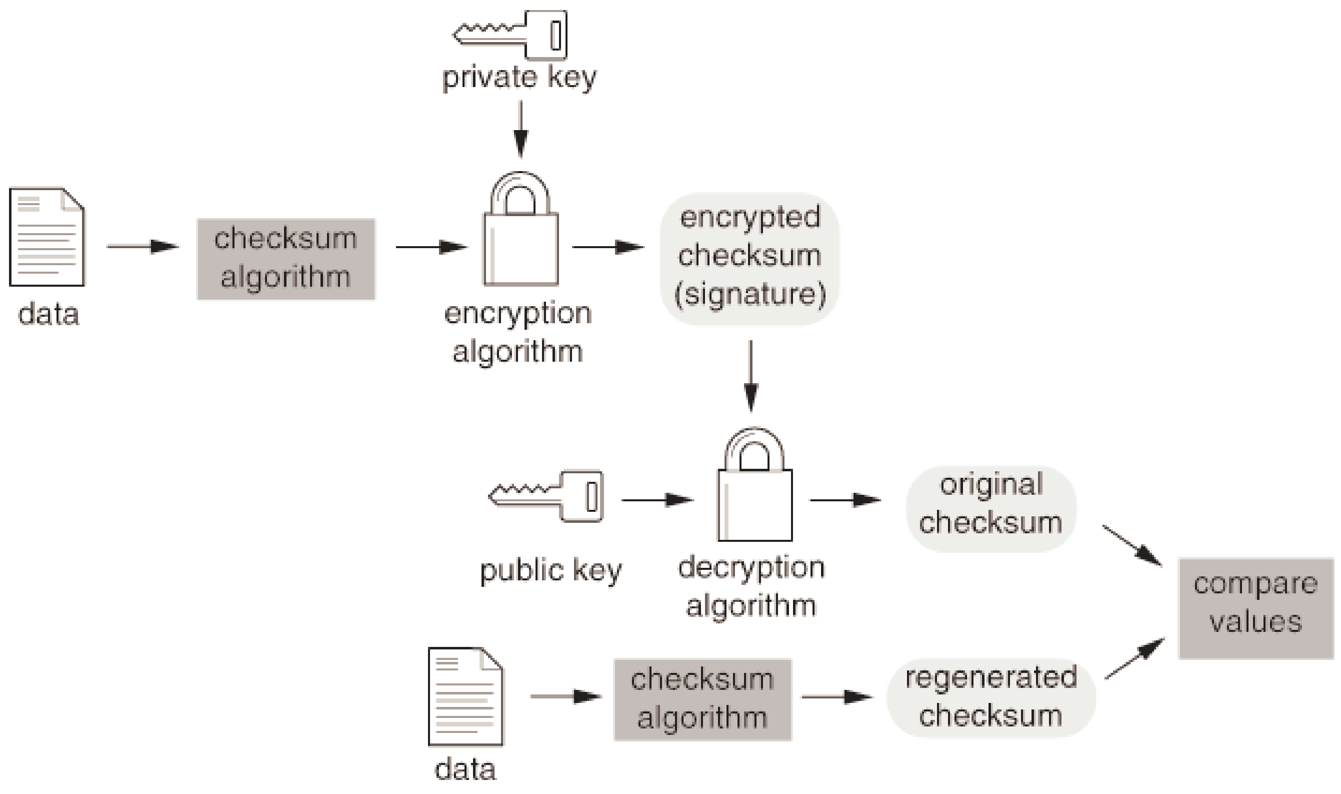
If you have the public key, you can encrypt messages to a client that possesses the matching private key. You cannot use the public key to decrypt messages that have been encrypted with the public key. Only the private key can decrypt the messages encrypted with the public key. Since only one person 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.

This application of public key encryption is the method used for creating a digital signature for data. The primary application of public key encryption in the NDS™ architecture is in Authentication, and is covered in detail later.

The following figure shows how asymmetrical encryption is used to create a digital signature. Note that in this example, only the checksum is encrypted. The encrypted checksum is sent along with clear-text (unencrypted) data.

Figure 11. Asymmetrical Encryption and Digital Signatures



NDS uses a combination of symmetrical and asymmetrical encryption to create a procedure that is both practical and secure.

Related Topics:

Public and Private Key Pairs

Authentication

Equivalence in NDS

It is possible for an object to be associated with other objects listed in an ACL. The association of one object with another for security purposes is called a **security equivalence**.

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.

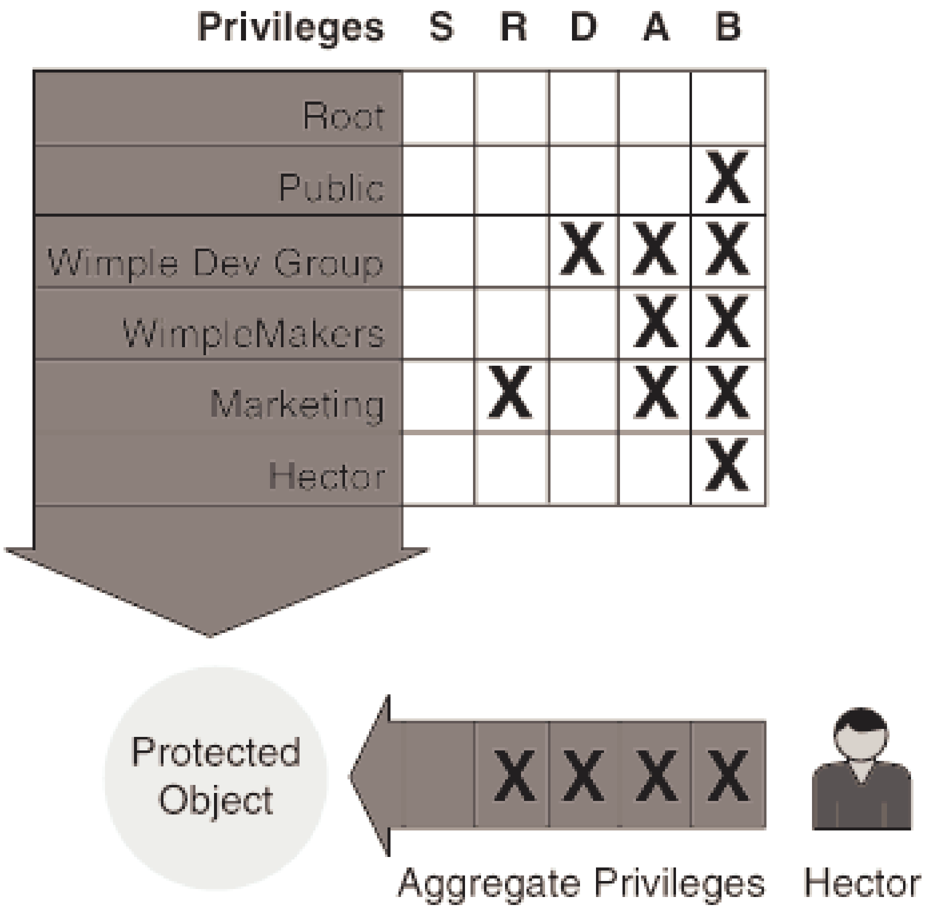
An object can be made security equivalent to a group object. Using the NetWare® utilities an object is made security equivalent to any group it is made a member of. Simply being a member of a group, however, does not give an object privileges of the group. It is having the group in the object's security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

Security equivalences are also formed by virtue of a subject 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 [Root]. This method uses the [Root] name as the subject of the privilege set. All objects are subordinate to the [Root] object by default so any privileges assigned to [Root] are applied to all objects. Similarly all NetWare 4.x objects are implicitly equivalent to [Public]. Even users not logged in to the NetWork (not authenticated) can be given privileges to objects in NDS by adding an ACL entry with [Public] as the subject of a privilege set.

The following figure shows how the accumulated privileges for Hector are determined regarding a particular protected object. Hector is the equivalent of 4 objects: [Public], Wimple Dev Group, WimpleMakers, and Marketing. Note that he is equivalent to WimpleMakers and Marketing by virtue of his being subordinate to these objects in NDS. He is equivalent to [Public] by definition and to Wimple Dev Group because his Security Equivalents attribute includes them (though this graphic does not show it). Even though no privileges are assigned explicitly to Hector, his aggregate privileges are the sum of all of his equivalences. Hectors effective privileges will be his aggregate privileges as long as no inheritance filters restrict them.

Figure 12. Security Equivalences



Related Topics:

NDS in Security Applications

NDS Security: Concepts Guide

Expression Filters for NDS Searches

You can build the search expression tree using the filter management functions. A search expression is formed from sub-expressions that can be linked by the logical nodes OR and AND. Each sub-expression, which might or might not be enclosed in parenthesis, takes the form: `(attribute name token CONDITION attribute value)`. The condition is usually a relational node such as EQ, GE, LE, or APPROX.

Let's look first at a simple example. The following expression sets up a search for objects with a Device attribute with a value equal to "toaster".

```
Device EQ "toaster"
```

The next example builds a search expression that looks for user objects with a Surname attribute not equal to "brown".

```
NOT(Surname EQ "brown")
```

Given the variety of tokens available, expressions can become quite complex, as shown below:

```
(Object_class EQ "user" AND Title EQ "department head") AND
(Location EQ "provo" OR Location EQ "orem")
```

Tokens are the basic building blocks that convert search expressions into search filters. A list of possible tokens follows. The token definitions can be found in NWDSFILT.H.

```
FTOK_END          0    // Terminates the expression tree.
FTOK_OR           1    // Link comparison statements
FTOK_AND          2
FTOK_NOT          3
FTOK_LPAREN       4    // Group comparisons and relational arguments
FTOK_RPAREN       5
FTOK_AVAL         6    // Allows an entry to represent a value
FTOK_EQ           7    // Logical operators
FTOK_GE           8
FTOK_LE           9
FTOK_APPROX       10   // Approximate actually is a string matching oper
FTOK_ANAME        14   // Associates an attribute name with a value
FTOK_PRESENT      15   // Tests if attribute value is assigned
FTOK_RDN          16   // Used to search for an object without authentic
FTOK_BASECLS      17   // Used to search for a class without authenticat
FTOK_MODTIME      18   // Returns objects modified since this time
FTOK_VALTIME      19   // Returns objects with this value that have been
                       // since this time
```

Once you form a search expression, you can build the expression tree of the search filter using **NWDSAllocFilter** and **NWDSAddFilterToken**. **NWDSAddFilterToken** adds a token to the search filter. Each token represents a node on the expression tree. If the token is an attribute name or an attribute value, a syntax ID is required. For example, a CN (Common Came) attribute or value is associated with the *CIString* (Case Ignore String) syntax. The following code segment demonstrates how to get the syntaxID:

```
retcode = NWDSGetSyntaxID(context,attribute-name, &syntaxID);
NWDSAddFilterToken(FilterCursor, token, value, syntaxID);
```

If the token is neither an attribute name nor an attribute value, the third and fourth parameters should be NULL, as follows:

```
NWDSAddFilterToken(cur,token,NULL,NULL);
```

NWDSAllocFilter allocates a filter expression and initializes the cursor.

The following code segment demonstrates how to build an expression into an expression tree.

```
Expression:
   NOT(Surname EQ "brown')

Code Example:
   NWDSAllocFilter(&FilterCursor);
   NWDSAddFilterToken(FilterCursor, FTOK_NOT, NULL,NULL);
   NWDSAddFilterToken(FilterCursor, FTOK_LPAREN, NULL,NULL);
   NWDSGetSyntaxID(context ,"surname",&syntaxID);
   NWDSAddFilterToken(FilterCursor, FTOK_ANAME,"surname', syntaxID);
   NWDSAddFilterToken(FilterCursor, FTOK_EQ, NULL,NULL);
   NWDSAddFilterToken(FilterCursor, FTOK_AVAL,"brown", syntaxID);
   NWDSAddFilterToken(FilterCursor, FTOK_RPAREN, NULL,NULL);
   NWDSAddFilterToken(FilterCursor, FTOK_END,NULL,NULL);
```

After you have added all of the filter tokens to the cursor, your next task is to store the filter expression into the filter buffer.

```
NWDSPutFilter(context, filterBuffer, FilterCursor, FreeValuePointer);
```

The *FreeValuePointer* parameter can be passed either NULL or a pointer to a function that frees the attribute values.

Related Topics:

Search Parameters

Searching NDS

Giving Rights in NDS

The flexibility and power of NDS security is due not only to the rights listed above, but in how they can be given to other objects. An object's rights can come through five basic methods.

Explicit Rights

Explicit rights are those rights specifically given the user to an object or its properties.

Inherited Rights

When a user is given [Object Rights] or [All Property Rights] to a container object, the user is given the same rights to all of the subordinate 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.

Note that if the user is given specific property rights, and not [All Property 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 property of a container, the user does not have Read rights to the Name property of objects subordinate to the container. However, if the user has [All Property Rights] to the container, the user has [All Property Rights] to all objects subordinate to that container as well.

Inherited Rights Filters

Inheritance of rights adds significantly to the flexibility and ease of administration in NDS security. However, it is sometimes necessary to stop the inheritance of certain rights. For example, suppose user Joe is given [Object Rights] of Browse, Create, and Rename to a container. This particular container has four subordinate container objects, each of which has other subordinate objects. We want Joe to inherit his rights from the first container down to all of the subcontainers except one. In this one subcontainer, we don't want Joe to be able to create subordinate objects. We're in a mess if we don't have a way to filter out Joe's [Object Rights].

This is where **Inherited Rights Filters** (IRFs) come in. We can place an IRF on the subcontainer where we want to filter the rights coming down. If we place an IRF at this container that filters out the Create [Object Rights], then all objects that have the Create [Object Rights] to the parent of this container, do not inherit the Create right to this container.

The IRF actually shows up in the ACL for the object. The Property is either [Object Rights] or [All Property Rights]. The Rights part of the ACL entry specifies which rights are allowed to be inherited. Any rights not listed in the ACL cannot be inherited.

Security Equivalence

Security Equivalence is an easy way to grant rights to users or other objects. When a user (in this case, a user object only) is made Security Equivalent to another object, that user has the same rights to the same objects as the object the user was made Security Equivalent to.

Every user object is made Security Equivalent to [Public]. The user is also Security Equivalent to each of the objects in its Security Equals property. Lastly, each time a user is added to a group, the user is made Security Equivalent to that group.

Although the [Public] Security Equivalence is a special case, all other Security Equivalence creates backlinks to other objects in the NDS tree that must be resolved each time NDS needs to calculate the effective rights of an object. Resolving backlinks involves lookups of objects in the tree that aren't cached on partition boundaries. This creates network traffic on the wire. Also, NDS is constantly verifying backlinks in the background to make sure that any object changes in other parts of the tree are reflected in the Security Equals property of user objects. As more users are assigned Security Equivalence, more network traffic is generated, and NDS must do more background processing to maintain the backlinks.

Containment

Containment, although similar in some ways to Security Equivalence, is much more efficient with network resources. Containment basically means that all objects in the NDS tree are automatically assigned the same rights as all objects that appear in their fully distinguished NDS names. In plain language, this means that every user has the same rights as each of its parent containers. So the user Joe.Sales.MyCompany has the same rights as Sales and MyCompany. Whatever rights are assigned to the Sales and MyCompany containers are inherited by Joe. This is pretty simple. What is not as obvious is that Joe also inherits the rights assigned to [Root]. The [Root] object is merely a container that holds all objects in the tree. Since [Root] is always there, it is not necessary to specify it in the distinguished network name.

Using containment effectively causes much less network traffic than does Security Equivalence because the information used in containment is cached on partition boundaries, and does not create the backlink maintenance that must be handled by NDS.

Related Topics:

Effective Rights Calculations

Reading Effective Rights: Example

NDS Security: Concepts Guide

Graphical View Explanation

The **Graphical View of NDS Object Class Definitions** module provides a visual representation of the Base Schema. It shows the object classes in the structure of the class hierarchy, rather than in alphabetical order presented by the text in NDS Object Class Definitions. The purpose of this appendix is to provide a visual view of the object classes, super classes, and inheritance.

Each object class is represented by a box that contains the name of the object class, its immediate super class, and a listing of the containment rules and attributes that are defined for that object class. This view does not show the default ACL templates that are listed in NDS Object Class Definitions, nor does it provide comments about the attributes.

In this view, the object classes that are placed above an object class are its super classes. The object classes that are below an object class are its subordinates. An object class inherits the rules and attributes defined by all of its super classes, the arrows show the direction of flow for inheritance.

Objects inherit from their super classes but they do not inherit from their subordinates. For example, Device (Class) inherits from Top, but it does not subordinates. For example, Device (Class) inherits from Top, but it does not inherit from Computer. However, Computer inherits from Top and from Device.

Each object class has the following information listed:

Super Classes. Objects of the class inherit information types and attributes from classes listed here. All object classes must have one or more super classes, except Top, which is a super class to all classes. (In this view,Top is shown multiple times, since it is a super class to all classes.) These listings just list the object's immediate super class.

Containment. Objects of the class may be created as subordinates in the NDS tree to objects of the classes listed here. An object of the class may not be subordinate to any object that is of a class not listed here.

Named By. The partial name or Relative Distinguished Name (RDN) of objects of the class consists of at least one of the attributes listed here. These attributes can be either mandatory or optional attributes, but at least one must be given a value when creating an object of the class. If the only*Named By* attribute is optional, it is in effect mandatory.

Mandatory and Optional Attributes. All attributes are either mandatory or optional. If an attribute is mandatory, a value must be assigned to that attribute. If an attribute is optional, an assigned value is not required, unless it is the only *Named By* attribute.

The Containment classes and **Named By** attributes of a class comprise a set of structure rules that define an object's relation to other objects in NDS. An object's RDN is determined by its Named By attributes; its distinguished name is determined by the objects it is subordinate, to which must be of a class found in the Containment classes. Hence, structure rules effectively control the formation of the distinguished names.

Related Topics:

NDS Object Class Definitions

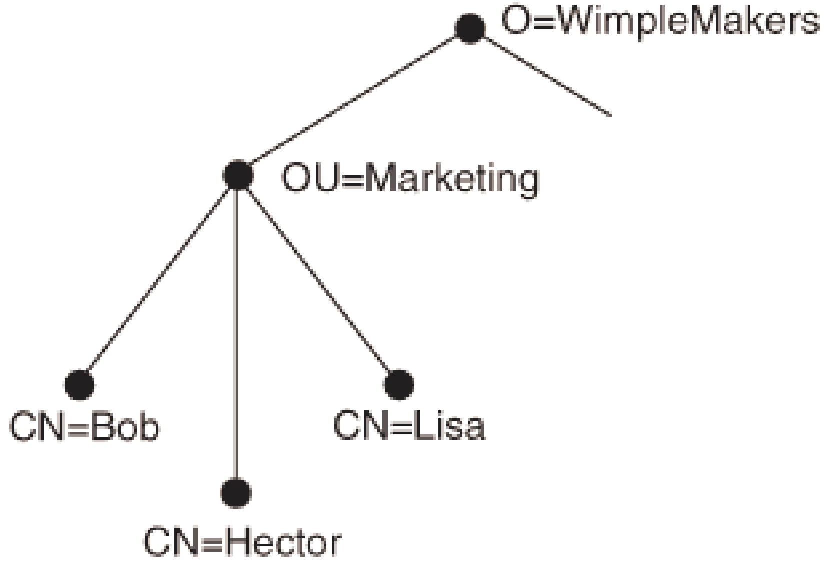
NDS Attribute Type Definitions

Graphical View of NDS Object Class Definitions

Hierarchical Naming

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

Figure 13. Example NDS Tree


O=WimpleMakers
OU=Marketing
CN=Bob
CN=Lisa
CN=Hector

WimpleMakers is actually subordinate to an object called [root] All NDS 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 NDS 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 NDS.

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 partial 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.

NDS 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 shows in the following example:

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

Related Topics:

Typeless Names

Attribute Type Abbreviations

NDS Context

NDS Context: Guides

Inheritance

Suppose you wanted to create a Toaster object class and give it attributes that apply only to toasters. You could assign your Toaster object class to the Device class for inheritance purposes. The Device class contains attributes that would be relevant to any device on the network, such as computers, printers, modems, or even toasters. The Toaster class would inherit the attributes of the Device class.

Here is a partial listing of the attributes of the Device object class: CN (Common Name), Object Class, ACL, Description, Locality Name, Network Address, and Serial Number. Two of these attributes, Object Class and ACL, were inherited from Top, which is the super class of Device. The Top object class contains attributes that are relevant to all objects on the network.

The principle of inheritance from super classes saves you from the task of redefining every attribute that your class needs. You are free to define only those attributes that are unique to your new class.

Related Topics:

Class Inheritance Rules

Super Class Structure

Inheritance in NDS

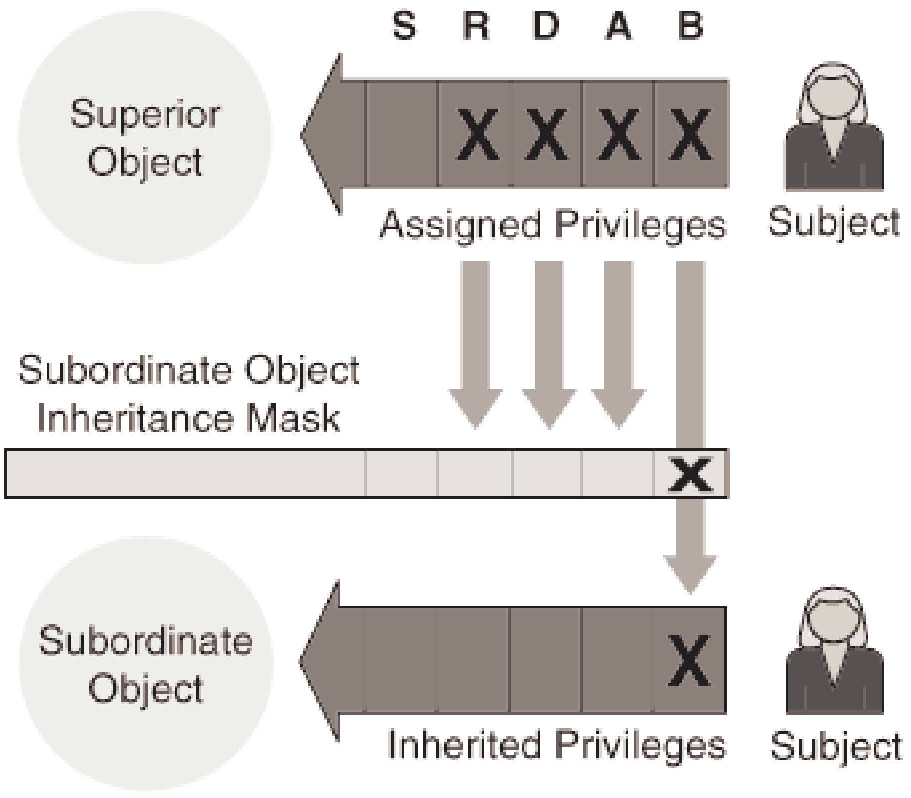
Access control privileges are applied according to the hierarchical structure of NDS. For example, if a subject has been granted the Browse privilege for an object, the subject will also have the privilege to Browse that object's subordinates. Applying a particular trustee privilege set to the protected object's subordinates is called **rights inheritance**.

Privileges assigned at the object level can be inherited in this manner. Trustee assignments made to **all attributes** can be inherited as well. However, trustee assignments given for individual attributes do not flow down the tree to subordinate object's attributes.

Inheritance can be restricted by Inheritance Masks. 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 privileges to one object that is the superior of another object. The subject has not been assigned privileges (R, D, A, and B) 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. If the inheritance bit is set the privilege is inheritable.

Figure 14. Inheriting Access Privileges



An inheritance mask applies only to inherited rights. 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. Assigning an inheritance mask to an object or attribute is optional. If no inheritance mask exists all inherited privileges are granted.

Related Topics:

Inheritance Masks

NDS Security: Concepts Guide

Inheritance Masks

Inheritance Masks control the inheritance of privileges. This implies that Inheritance Masks are used to selectively filter privileges down the NDS tree. Inheritance Masks are stored as ACL values of NDS objects. NDS uses this mask to compute the effective rights one object has on another object or attribute.

Inheritance Masks can be defined for objects, individual attributes, or all attributes of an object. Inheritance Masks are optional. 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 **Subject Name** of the ACL value.

Related Topics:

Rights to an NDS Object

Inheritance in NDS

NDS Security: Concepts Guide

Initialization Operations for NDS Buffers

Value	Operation Type
1	DSV_RESOLVE_NAME
2	DSV_READ_ENTRY_INFO
3	DSV_READ
4	DSV_COMPARE
6	DSV_SEARCH
7	DSV_ADD_ENTRY
8	DSV_REMOVE_ENTRY
9	DSV_MODIFY_ENTRY
10	DSV_MODIFY_RDN
11	DSV_DEFINE_ATTR
12	DSV_READ_ATTR_DEF
13	DSV_REMOVE_ATTR_DEF
14	DSV_DEFINE_CLASS
15	DSV_READ_CLASS_DEF
16	DSV_MODIFY_CLASS_DEF
17	DSV_REMOVE_CLASS_DEF
18	DSV_LIST_CONTAINABLE_CLASSES
19	DSV_GET_EFFECTIVE_RIGHTS
20	DSV_ADD_PARTITION
21	DSV_REMOVE_PARTITION
22	DSV_LIST_PARTITIONS
23	DSV_SPLIT_PARTITION

24	DSV_JOIN_PARTITIONS
25	DSV_ADD_REPLICA
26	DSV_REMOVE_REPLICA
27	DSV_OPEN_STREAM
28	DSV_SEARCH_FILTER
31	DSV_CHANGE_REPLICA_TYPE
38	DSV_SYNC_PARTITION
39	DSV_SYNC_SCHEMA
40	DSV_READ_SYNTAXES
41	DSV_GET_REPLICA_ROOT_ID
42	DSV_BEGIN_MOVE_ENTRY
43	DSV_FINISH_MOVE_ENTRY
44	DSV_RELEASE_MOVED_ENTRY
45	DSV_BACKUP_ENTRY
46	DSV_RESTORE_ENTRY
50	DSV_CLOSE_ITERATION
53	DSV_GET_SERVER_ADDRESS
54	DSV_SET_KEYS
55	DSV_CHANGE_PASSWORD
56	DSV_VERIFY_PASSWORD
57	DSV_BEGIN_LOGIN
58	DSV_FINISH_LOGIN
59	DSV_BEGIN_AUTHENTICATION
60	DSV_FINISH_AUTHENTICATION
61	DSV_LOGOUT
62	DSV_REPAIR_RING
63	DSV_REPAIR_TIMESTAMPS
69	DSV_DESIGNATE_NEW_MASTER
72	DSV_CHECK_LOGIN_RESTRICTIONS

Related Topics:

DEFAULT_MESSAGE_LEN Constant

Preparing NDS Input Buffers

NDS Buffer Management Introduction

Introduction to the NDS Schema

NDS™ is governed by a set of rules that define the types of objects that can exist in an NDS tree. This set of rules is called the **Schema**.

The schema defines how objects in the NDS database are created and managed through the following information types:

Object Classes, which describe the types of objects that can exist in the database

Attribute Types, which provide particular information in an object class

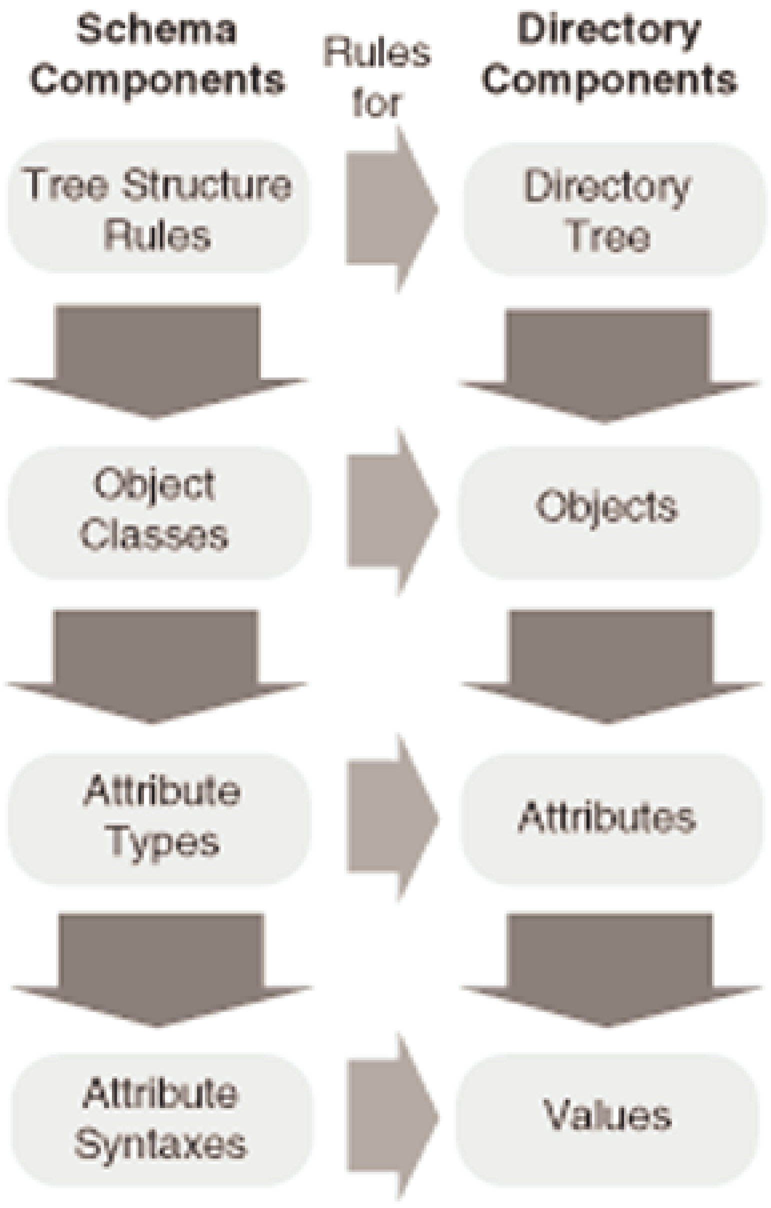
Attribute Syntaxes, which define the type of data stored in an attribute type

Each object belongs to an object class that specifies what attributes can be associated with the object. All attributes are based on a set of attribute types that are, in turn, based on a standard set of attributes syntaxes.

NDS has a set of built-in classes and attribute types that accommodate general categories of network objects such as organizations, users, and devices. This set is called the base schema. As a developer, you can build on the base schema to create new classes for specific kinds of objects.

The NDS Schema not only controls the structure of individual objects, but it also controls the relationship among objects in the NDS tree. The Schema rules allow some objects to contain other subordinate objects. Thus the Schema gives structure to the NDS tree.

The figure below shows how the Schema components and the NDS components are interrelated. The vertical arrows indicate the structure dependencies from the basic building blocks up to the NDS Schema and the NDS tree, respectively. The horizontal arrows denote the Schema rules that apply to the respective NDS components.



Related Topics:

NDS Schema Components

Synchronizing with NDS Database - part 1: Example

Synchronizing with NDS Database - part 2: Example

Synchronizing with NDS Database - part 3: Example

Lists Stored in NDS

NDS is a convenient place to store lists. A list is associated directly with a specific object. Typically these lists contain names of other objects in NDS. 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. NDS provides several standard lists for specific objects, such as users and servers. Applications can also define lists for their own special needs.

Related Topics:

Descriptions Stored in NDS

Types of Information Stored in NDS

Logical Operators for NDS Searches

AND, OR, and NOT are logical operators. These tokens express the logical relationship among attribute value assertions. You can control precedence among the logical operators by inserting tokens that act as parentheses. In the absence of parentheses, the AND operator takes precedence over the OR operator, and the NOT operator takes precedence over both.

The following table shows the logical operators and the conditions that test TRUE for each.

Table auto. Logical Operators

Token	Value	Comment
FTOK_OR	1	TRUE if either subordinate node is true.
FTOK_AND	2	TRUE only if both subordinate nodes are true.
FTOK_NOT	3	TRUE if the node is false.

FTOK_LPAREN	4	Left parenthesis.
FTOK_RPAREN	5	Right parenthesis.

Related Topics:

Search Expression Trees

NDS Search Introduction

Lookup in NDS

Lookup is probably the most obvious approach to searching NDS. The user identifies a particular object and NDS 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, NDS 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 NDS API.

Related Topics:

Browsing NDS

Retrieval of Information from NDS

Mandatory and Optional Attributes

The attributes assigned to a class can be either **mandatory** or **optional**. If an attribute is mandatory, you must assign a value. An optional attribute does not require a value, but you are not required to do so unless the attribute is the only attribute used for naming the object.

A client cannot associate an attribute with an object if the attribute is not listed among the mandatory or optional attributes of the object's expanded base class definition. If a client needs to associate an attribute with a particular object and the attribute is not specified by the object class, then the client must either add the attribute to the class as an optional attribute or define a new class that inherits from the original class and includes the additional attribute.

Suppose you wanted to create a Toaster object in your new Toaster class. You would have at least two mandatory attributes: CN (Common Name), inherited from the Device class; and Object Class, inherited from the Device class, which, in turn inherited from Top. Additionally, you could make use of any or all of the 22 optional attributes your Toaster class inherited.

Related Topics:

Attribute Type Information

NDS Schema: Guides

Mandatory Partition Attributes

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

Attribute	Comment
Convergence	This attribute determines how frequently the partition attempts to synchronize its replicas.
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.

Related Topics:

Optional Partition Attributes

Partition Information

Multiple Tree Functions

Ideally, NDS would provide us with one globally available information tree. The tree would be multicorporate and multinational. Once we established a connection to the global tree, we could access information and services anywhere in the world, as long as we had the proper access rights.

The current reality, however, is not a single global tree, but a forest of independent trees. Within many large organizations, a user may be required to access several trees to accomplish an assigned task. For this reason, NDS must support access to multiple trees.

The following functions provide NDS support for connecting to multiple trees.

NWDSAuthenticateConn

NWDSCanDSAuthenticate

NWDSGetDefNameContext

NWDSGetMonitoredConnRef

NWDSOpenMonitoredConn

NWDSScanConnsForTrees

NWDSScanForAvailableTrees

NWDSSetDefNameContext

NWDSReturnBlockOfAvailableTrees

Related Topics:

NDS Search Introduction

NDS Access: Guides

Name Caching

An enhancement was made to the resolve name functionality that is required for every request of the NDS engine and should deliver substantial performance gains across WAN links and NetWare Connect.

The **NWDSSetContext** and **NWDSGetContext** functions can be called using DCK_NAME_CACHE_DEPTH to query or set the depth of the name cache (how many names the cache will remember for the context handle).

If the depth of the name cache is decreased, the current cache will not be affected. If the depth of the name cache is set to zero, the current cache will be completely cleared.

An effective way to clear the name cache is to call the **NWDSSetContext** (context, DCK_NAME_CACHE_DEPTH, &0) function followed by calling the **NWDSSetContext**(context, DCK_NAME_CACHE_DEPTH, &5) function again.

Currently, the default depth of the name cache is five.

Related Topic:

NDS Context: Guides

Name Expansion

Names sent to NDS must also be distinguished names. What that means is that any partial names passed in must be translated to DNs. The libraries do this for you by appending the current context to any name that you pass in. For example:

```
Name passed in:       JRoss
Current Context:      HR.ACME
Resulting Name:       JRoss.HR.ACME
```

So what happens if you pass a name that is already a Distinguished Name? Like the Default Typing Rule, there are also rules for expanding names.

Name Expansion Rules

A period preceding a name prevents the libraries from appending the context to the name.

For each trailing period, the libraries remove one component from the context before appending it to the name.

If you place a period at the beginning of the name passed in, the libraries treat it like a distinguished name and do not append the context to the end. Here is an example:

```
Name passed in:       .Ppearson.Engineering.Pub.ACME
Current Context:      Payroll.HR.Pub.ACME
Resulting Name:       Ppearson.Engineering.Pub.ACME
```

For each period you place at the end of a name, the libraries remove a naming component from the context before appending the context to the name you passed in. For example:

```
Name passed in:       Ppearson.Enginnering..
Current Context:      Payroll.HR.Pub.ACME
Resulting Name:       Ppearson.Engineering.Pub.ACME
```

Note that in the two examples the same person is being referenced and both examples use the same context. Also note that in the first example the name passed in was 30 characters and the second name pass in was only 22 characters. Both examples resulted in the same name but used different rules to achieve that name.

Related Topics:

Alias Naming

NDS Context: Guides

Name Functions

The following table shows the NDS functions that are used for performing operations on NDS names.

Function	Purpose
NWDSAbbreviateName	Converts a name to its shortest, typeless form relative to a specified name context

NWDSCanonicalizeName	Converts an abbreviated name to the canonical form
NWDSRemoveAllTypes	Removes all attribute types from a distinguished name.

Related Topics:

NDS Context

NDS Context: Guides

Name Rules

Objects are identified by their own names and the names of their parent objects. The name of an object is called its **partial name** or **Relative Distinguished Name (RDN)**. For example, a user's partial name might be the following.

```
CN=Fred
```

The full name of an object, which includes the names of its parent objects, is called the **complete name** or **Distinguished Name (DN)**. For example, a user's complete name may be the following.

```
CN=Fred.OU=Client.OU=Engineering.O=Novell
```

One or more attributes are specified for each class to be used in naming objects of that class. These are the only attributes that can serve as part of the partial name for objects of that class. For example, Organization objects are named by the *O (Organization)* attribute. This is the only attribute value that can appear in the object's partial name. Naming attributes can be designated as mandatory or optional, but if an optional attribute is the only one designated in the naming attribute, in effect, it is mandatory.

Naming attributes are listed under the "Named By" headings in the object class definitions found in NDS Object Class Definitions.

A naming attribute is not necessarily a reflection of the class to which an object belongs. Many classes, such as Computer, User, and Server, are named by their *CN (Common Name)* attribute. In such names, the name attribute itself gives no indication as to which class the object belongs. (The value of the name attribute might suggest the nature of the object.) On the other hand, some naming attributes are closely tied to specific classes. For example, the *C (Country Name)* attribute is used to name Country objects.

Related Topics:

Super Class Structure

NDS Names

Structure Rules

Name Server Information

Physically, a name server is a network node that administers zero or more NDS replicas. NetWare® servers are defined in NDS as NCP Server objects. A name server is defined as an NCP Server that provides name services for an NDS tree. All NetWare 4.x servers are also name servers. If an NCP Server is also a name server, a name server pseudo-object is created on that server. The pseudo-object is not accessible to clients. It stores the following system information that support NDS.

Server name

Network addresses

The network address of a name server is loaded by the AUTOEXEC.NCF when the server comes up. The name server at this address is responsible for updating this information if it changes addresses.

Related Topics:

Tree Walking

NDS Partition: Guides

Names Stored in NDS

The most significant piece of information associated with an object is its **name**. A name is identification for an object within the context of NDS. 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 NDS. A person using NDS 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,NDS makes it easy for people using NDS to obtain less intuitive kinds of information.

Related Topics:

Addresses Stored in NDS

Types of Information Stored in NDS

NDS Agents

NDS is responsible for managing the information stored in NDS and coordinating distributed operations with other servers. It manages all NDS requests, including those to access NDS, manage partitions, manage the NDS Schema and perform access control, perform authentication, and do time synchronization between servers.

NDS include synchronization of replicated information and performing searches of the NDS information upon request.

Related Topics:

NDS Bindery

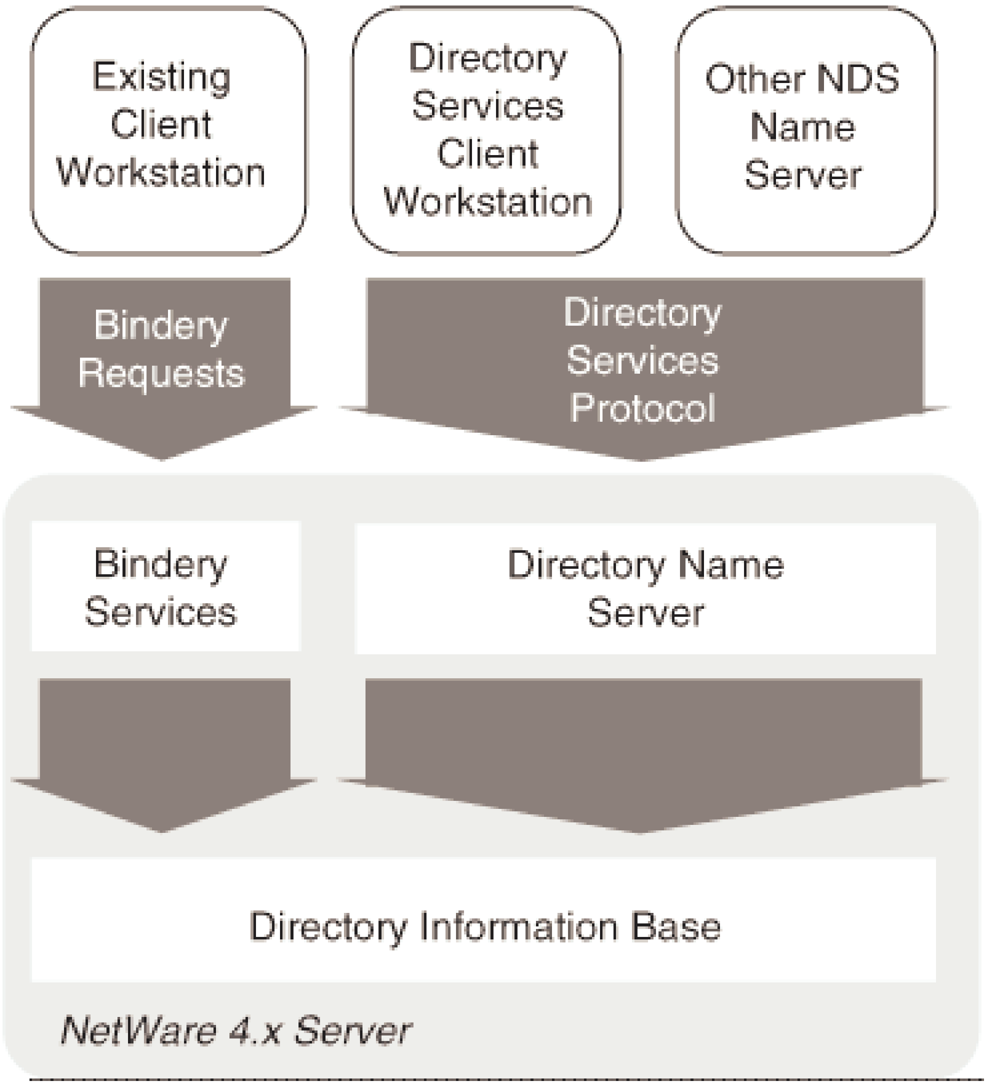
NDS Server Components

NDS Architecture Introduction

At the workstation, a user application invokes a **client agent** that formulates directory requests on the user's behalf. The client agent establishes a session with a **NDS Server** to which it submits the requests.

The client agent submits directory requests via the NDS Protocol. NDS servers also use this protocol when they need to make access requests. Existing workstations can access the information stored in NDS through NDS Bindery and require no modifications. The following figure shows the client/server architecture for NDS.

Figure 15. NDS Client / Server Architecture


Existing Client Workstation
Directory Services Client Workstation
Other NDS Name Server
Bindery Requests
Directory Services Protocol
Bindery Services
Directory Name Server
Directory Information Base
NetWare 4.x Server

The NDS database is called the Directory Information Base. NDS is a collection of objects with associated attributes. An object in NDS is specified according to the object's position in the NDS tree, the hierarchical tree of names in NDS. To inquire about an object, a user must determine the naming path of the object from the root of the tree to the object's name and submit the path to an NDS server.

NDS receives and processes the request, and returns the results. NDS is part of the NetWare® core operating system. The Directory Information Base is implemented on NetWare 4.x servers using a native NDS record manager.

An NDS server is a NetWare server that provides NDS. All NetWare 4.x servers provide NDS.

Related Topics:

NDS Workstation Components

NDS Server Components

NDS Architecture: Concepts Guide

NDS Access and the File System

The access controls and restrictions to NDS are similar in concept to those used for access to the NetWare File System. This presents a potential confusion on the issues of NDS security versus File System Security.

First of all it is important to note that the security controls for accessing the file system are part of the file system not part of NDS.

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

Have NDS privileges to see and locate the server and volume

Establish a licensed connection to the server.

Have file system access privileges

In order to locate the portion 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 fileservers in the network will be represented by an object in NDS. If you have NDS 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. Obtaining approval to access a server based on NDS access privileges is called authentication.

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 NDS then requesting a connection.

If you have Write or Supervisor privileges on a server object in NDS you automatically have Supervisor privileges on that server's file system. Other than that your privileges to the file system of 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 OS are the same as for previous versions of the OS. These trustee assignments are kept along side the volume with which they are associated. They are not part of a distributed database such as the NDS Information Base. There may be mirroring of this information as provided by the System Fault Tolerant™ (SFT™) feature of NetWare but this is different and separate from the concepts and implementation of the distributed database of NDS.

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.

NDS identifies volumes and the file system stores and enforces its own access controls.

Related Topics:

NDS Security Function

NDS Security: Concepts Guide

NDS Attribute Functions

You can perform three types of operations on attribute type definitions as follows:

You can create a new attribute type definition using **NWDSDefineAttr**.

You can read existing attribute type definitions using **NWDSReadAttrDef**.

You can delete attribute type definitions using **NWDSRemoveAttrDef**.

Related Topics:

NDS Schema Extensions

NDS Class Functions

NDS: Functions

NDS Schema: Guides

NDS Bindery

NDS emulates Bindery 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.

> **NOTE:** Bindery is limited. While it allows backward compatibility, Bindery cannot express all of the information stored in NDS.

Related Topics:

Directory Information Base

NDS Server Components

NDS Buffer Allocation and Initialization Functions

Function	Comment
NWDSAllocBuf	Allocates an Buf_T structure and the requested number of bytes.
NWDSInitBuf	Initializes an Buf_T structure for input.
NWDSFreeBuf	Destroys an Buf_T structure and frees the memory allocated to it.

Related Topics:

Preparing NDS Input Buffers

Preparing NDS Output Buffers

NDS Access: Guides

NDS Buffer Management Introduction

Applications access NDS™ services through functions that might require input and output parameters in a complex variety of data configurations. For example, some functions require input parameters of multiple objects with multiple attributes assigned multiple values. Other functions return parameters that are equally complex.

To accommodate the data requirements of the various functions and to provide a flexible client interface, most NDS functions pass input and output parameters through specially defined local buffers. Therefore, an understanding of buffer management is essential to developing NDS aware programs.

In general, buffer management involves the following steps:

1. Allocate any input and output buffers required by an operation.
2. Initialize any input buffers.
3. Place any input parameters that define your operation into the input buffer.
4. Execute the request.
5. After the operation is complete, retrieve any results from the output buffer.

6. Clean up by deallocating buffers you no longer need.

The following topics provide further detail about NDS buffer management:

Buffer Size in NDS

Initialization Operations for NDS Buffers

DEFAULT_MESSAGE_LEN Constant

Buffer Allocation Types and Related Functions

Related Topics

Preparing NDS Input Buffers

Preparing NDS Output Buffers

NDS Access: Guides

NDS Class Functions

There are essentially five operations you can perform on object class definitions.

You can create a new class using **NWDSDefineClass**.

You can modify an existing class using **NWDSModifyClassDef**.

You can read a class definition with **NWDSReadClassDef**.

You can delete a class with **NWDSRemoveClassDef**.

Or you can list all of the containable classes using **NWDSListContainableClasses**.

Related Topics:

Attribute Types and Attribute Syntax

NDS Attribute Functions

NDS: Functions

NDS Schema: Guides

NDS Compliance to X.500 Standard

Directory services are information providers. To make the services provided by the directory useful, applications must have standard ways to access and interact with them. Toward this end, Xerox* created the X.500 Directory

Services Standards.

X.500 differs from other earlier directory service designs and implementations in three main ways. First, X.500 specifies a naming and addressing structure, which functions at a global level; that is, across multiple organizations. Second, X.500 specifies how directory service information should be provided not only to applications, but also to users. And finally, X.500 supports extensive query techniques that go far beyond standard name-to-address mapping functions.

NDS incorporates and, in its implementation, actually surpasses the X.500 standards. As such, any software designed to interact with an X.500 directory service will interact with NDS. NDS also provides many extra functions that software users and designers can take advantage of in their own applications.

Related Topics:

Benefits of NDS

Introduction to NDS Development: Guide

NDS Context

When dealing with NDS, a **context** is used as a reference point into an NDS tree. The complete name of the container pointed to is called the **name context**, and serves as a default naming path for NDS operations. When a workstation shell is booted, the name that is used for the context comes from the client networking configuration files. (On a DOS/Windows machine this is known as the NET.CFG file).

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 distinguished 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 do 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.

You might ask, "Why not always use the DN?" If you are dealing with thousands of users and they are all several layers down in the tree, you are going to use a lot of memory if you are using the distinguished names. The bottom line is this: be careful about storing RDNs for later use if you are going to be changing the current context.

The name context is implemented as a value in the operation's NDS context variable that is created with **NWDSCreateContext**. 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 NET.CFG 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 a NDS 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 NDS functions.

Note that **the NDS functions do not modify the NET.CFG or STARTUP.NCF files**.

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. 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
```

The name context also provides one method for processing typeless names. The sequence of types in the name context can serve as a default sequence that the client agent applies to a typeless name. However, the effect of a name context on input name is controlled by the client through use of context qualifiers. The qualifiers control both the expansion of partial names into full name paths and the typing of typeless names.

Related Topics:

Context Key Values

Hierarchical Naming

Typeless Names

NDS Context: Guides

NDS in Security Applications

This module describes how you, as a software developer, can use NDS security in your applications.

First of all, some of the terminology changes when you go behind the scenes to the programmer's perspective. Here are some of the most important changes:

[Object Rights] is called [Entry Rights]

Properties are called Attributes

[All Properties Rights] is called [Attribute Rights]

Most of Security programming is very simple when you know how to read and write information about objects in the NDS tree. Reading and writing objects and their information is discussed in detail in other chapters, so we will focus here on what is specific to security.

Let's look at what a program would do to read the ACLs of a particular object. Once you become familiar with NDS programming in general as described in the other chapters, there's not much new to security. The following program is presented in pseudo-code to simplify our discussion:

```
NWCallsInit()           These first two functions just set up the DLLs
NWInitUnicodeTables()   for us to use.

NWDSCreateContext()     Get a context allocated for our program to
                        communicate with NDS

NWDSAllocBuf()          Allocate a buffer for the ACL results to be
                        stored in

NWDSAllocBuf()          Used to store attribute names in NWDSRead

NWDSRead()              infoType=1, allAttrs=FALSE, iterationHandle=-1

NWDSGetAttrCount()      How many attributes are in the buffer? Better
                        just be one!

NWGetAttrName()         We already know the name is ACL, but we need
                        the valCount

while (valcount--)      Loop through the buffer, and process each ACL
{
   NWDSComputeAttrValSize()   How big is our next attribute value?
   NWDSComputeAttrValSize()   How big is our next attribute value?

   OurMem=malloc()            Allocate memory to store the next value

   NWDSGetAttrVal()           Put the attribute value (our ACL) in the
                              buffer we just allocated memory for

   /* when we use the NWDSGetAttrVal() we'll be using a structure Objec
      which holds all of the ACL information we need. We can process th
      information to our heart's content, and let our application do wh
      it needs to */

   free(OurMem)               We need to keep memory clean!
}
NWDSFreeBuf()           Free each of the buffers we allocated earlier

NWDSFreeContext()       Free the context we alloced earlier

NWDSFreeUnicodeTables() Now we've cleaned up all the NDS housecleaning
```

You use the standard functions **NWDSModifyObject** and **NWDSRead** for reading and writing ACLs to objects. You will notice from the comments in this pseudocode that reference is made to an Object_ACL_T structure. This is the structure that holds the contents of the ACL. The structure type definition is:

```
typedef struct
{
 char               *protectedAttrName;
 char               *subjectName;
 NWDS_PRIVILEGES    privileges;
}Object_ACL_T;
```

Generally, you refer to specific attributes of an object when granting rights. As discussed earlier from the administrator's perspective on security, you can use special notations to refer to all object rights or all property rights. These notations are summarized below.

When you use [Public] as the subject name, you are, in effect, granting all users those rights.

When you use [Inheritance Mask] as the subject name, you are setting up an Inherited Rights Filter.

When you use [Entry Rights] as the attribute name, you are giving the rights to the object, as in [Object Rights] described earlier.

When you use [Attribute Rights] as the attribute name, you are giving (or reading) the rights to the entire set of attributes.

When you use [SMS Rights] as the attribute name, you are referring to the rights to backup the object.

Let's say that user Joe.Sales.MyCompany was given rights to all attributes of the printer object Printer1.Accounting.MyCompany. The *protectedAttrName* would be "[Attribute Rights]", which indicates Joe has rights to all properties, or attributes. The *subjectName* would be "Joe.Sales.MyCompany", to indicate the user that has the rights. And the*privileges*, which is a 32-bit value, would have the lower bits set according to the privileges granted to user Joe.

Related Topics:

Object Management in NDS

NDS Security: Concepts Guide

NDS Input Buffer Functions

Function	Comment
NWDSBeginClassItem	Begins the insertion of a class definition into an input buffer.
NWDSPutAttrName	Inserts the name of an attribute into an input buffer.
NWDSPutAttrVal	Inserts an attribute value into an input buffer.
NWDSPutChange	Inserts a change record into an input buffer.
NWDSPutClassItem	Inserts a class item into an input buffer.
NWDSPutClassName	Inserts the name of an object class into an input buffer.
NWDSPutSyntaxName	Inserts the name of a syntax into an input buffer.
NWDSPutFilter	Inserts a search filter expression tree into an input buffer.

Related Topics:

NDS Output Buffer Functions

Preparing NDS Input Buffers

NDS Access: Guides

NDS List Functions

Before you can extract information from an object, you must know what objects are available. That is the purpose of list functions. Essentially, list functions return a list of objects that meet certain conditions.

NWDSList

Lists the objects that are subordinate to a specified object.

NWDSListByClassAndName

Lists objects of a particular object class and/or object name that are subordinate to a specified object.

NWDSListContainers

Lists container objects that are subordinate to a specified object.

NWDSExtSyncList

Lists the immediate subordinates of an object placing restrictions on the names, classes, modification times, and object types of the subordinates.

Related Topics:

NDS Read Functions

NDS Access: Guides

NDS Introduction

A directory service is a specialized database that contains network information. The information might be the name of a user, the name of a printer, or where on the network that printer is located. The information might even be the phone number of the local pizza delivery. NDS™ is a database framework ready to contain any data you might want to store.

By storing network information in a single place and making it easy to access, NDS gives users the ability to get information anytime, anywhere.

Related Topics:

Types of Information Stored in NDS

Introduction to NDS Development: Guide

NDS Names

NDS™ names are made up of a series of components and their respective types. A typical NDS 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, which means common name, is the naming attribute for a user object. The same goes for Engineering---Engineering is the name of an Organizational Unit and OU is the naming attribute for an Organizational

Organizational Unit and OU is the naming attribute for an Organizational Unit. Each component in the name is separated by a period.

Related Topics:

Hierarchical Naming

Attribute Type Abbreviations

Default Typing Rule

NDS Context

NDS Context: Guides

NDS Output Buffer Functions

Function	Comment
NWDSComputeAttrValSize	Returns the size of the next attribute value in a result buffer.
NWDSGetAttrCount	Returns the number of attributes in a result buffer.
NWDSGetAttrDef	Returns the next attribute definition in a buffer.
NWDSGetAttrName	Returns the next attribute and the number of associated values in a result buffer.
NWDSGetAttrVal	Returns the next value of an attribute in a result buffer.
NWDSGetClassDef	Returns the next class definition in a result buffer.
NWDSGetClassDefCount	Returns the number of class definitions in a result buffer.
NWDSGetClassItem	Returns the next item of a class item list in a result buffer.
NWDSGetClassItemCount	Returns the number of items in a class item list in a result buffer.
NWDSGetObjectCount	Returns the number of objects in a result buffer.
NWDSGetObjectName	Returns the next object name in a result buffer.
NWDSGetPartitionInfo	Returns the next partition information structure in a result buffer.
NWDSGetServerName	Returns the name of the server and the number of partition names in a result

	buffer.
NWDSGetSyntaxCount	Returns the number of syntax definitions in a result buffer.
NWDSGetSyntaxDef	Returns the next syntax definition in a result buffer.

Related Topics:

Controlling Iterations

Preparing NDS Output Buffers

NDS Access: Guides

NDS Read Functions

When we speak of reading NDS objects, we are actually talking about retrieving the information that is stored mainly as attributes. For example, a User object could have attributes that include the user's Surname, Email Address, Title, Telephone Number, or maybe even an Obituary.

NWDSRead

Reads values from one or more attributes of the specified object.

NWDSReadObjectInfo

Reads object information not stored in the attributes of the object.

NWDSExtSyncRead

Reads values from one or more attributes of the specified object allowing restrictions on modification time.

Related Topics:

Configurable NDS Functions

NDS List Functions

Deleting a User Object: Example

NDS Access: Guides

NDS Schema Components

The NDS Schema consists of three basic components:

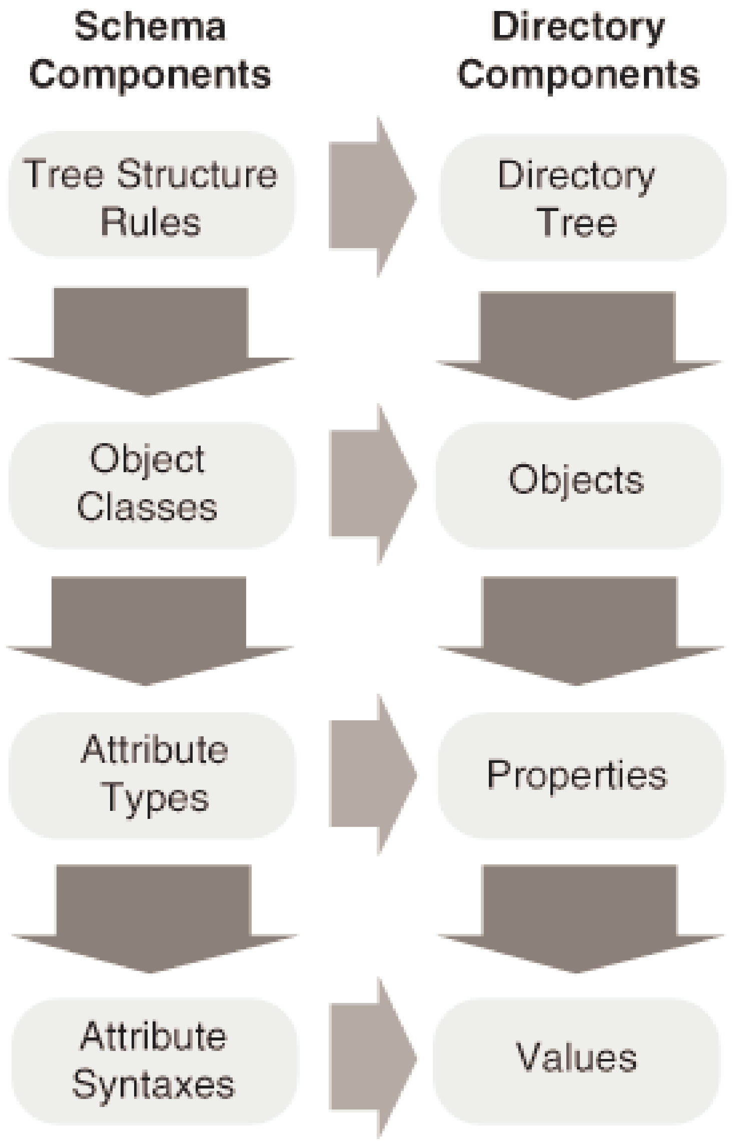
Object Classes (Object Classes in NDS)

Attribute Types (Attribute Types and Attribute Syntax)

Attribute Syntaxes (Attribute Types and Attribute Syntax)

The following figure shows how the components are related. Attribute syntaxes define the standard data types for values stored in NDS. Attribute types are constructed from the attribute syntaxes to define the categories of information that can be associated with objects. Object classes are constructed from the attribute types. Object classes also have structure rules, the constraints that control the construction and hierarchy of the NDS tree.

Figure 16. Schema Components



The Schema definitions themselves have a relationship with one another that could be referred to as a structure. This Schema structure is established through the following relationships:

- Objects to other objects
- Attributes to objects
- Syntaxes to attributes

The Schema structure is established primarily by the super classes in the object class definitions which are discussed in detail later.

NDS clients should note that each NDS name server stores a copy of the NDS Schema. Writable copies of the Schema are found only on those servers storing a replica of the **root** partition. The client agent automatically targets one of these servers for Schema modifications.

Changes to the Schema are propagated across NDS according to the synchronization schedule of each NDS server.

Related Topics:

- Object Classes in NDS
- NDS Schema: Guides

NDS Schema Extensions

The following list summarizes the operations you can perform on the NDS Schema:

- You can create, read, and delete attribute type definitions.
- You can create, read, and delete object class definitions.
- You can list containable classes.
- You can modify class definitions.
- You can read attribute syntax definitions.
- You can read attribute syntax IDs.

The only thing you might want to do, but can't, is to create or modify an attribute syntax. The attribute syntaxes are the most fundamental building blocks of NDS and cannot be altered.

The following sections of this chapter introduce you to the functions used to perform the operations in the preceding list.

NOTE: You must have Admin equivalent rights to extend the NDS schema. And, if you are working on a workstation client, you must initialize the Unicode tables.

Related Topics:

Distribution of the NDS Schema

Registering Attribute Types and Class Definitions

NDS Attribute Functions

NDS Class Functions

NDS Schema: Guides

NDS Search Introduction

Searching is somewhat more complicated than other NDS operations. This is because of the many factors affecting a search operation. A search request must include not only a search criteria, but also the area to be searched and the amount of information to be returned for each matching object.

The following topics will make NDS searches more understandable:

Buffers Needed for NDS Searches

Expression Filters for NDS Searches

Search Parameters

Search Filters

Relational Operators for NDS Searches

Logical Operators for NDS Searches

Search Expression Trees

Related Topics:

Buffers Needed for NDS Searches

NDS Access: Guides

NDS Security Function

There is only one function specific to security, **NWDSGetEffectiveRights**. By making a call to this function, you can determine the effective rights of one object to access another object or its attributes.

Related Topics:

NDS Security Introduction

NDS Security: Concepts Guide

NDS Security Introduction

NDS Security, which is also called **Access Control**, is extremely powerful and flexible. A network administrator can control access from any object in the NDS tree to any other object in the tree. Any two users---groups, containers, or any other NDS 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.

For the sake of clarity, User objects will be used in the examples throughout this chapter; however, security applies to almost any NDS object in the same way it applies to a user. Any NDS object can have the same rights to other objects as a user. Keep this in mind throughout this security discussion.

You need a good understanding of the following points about NDS Security:

NDS security is controlled through Access Control Lists (ACLs). The ACL of an object is an optional attribute of every object. It is assigned to the Top class, which means it is an inherited property to every class in the NDS tree, as every object class descends (inherits) from Top. The ACL has an entry for every other object that has rights to this object. For example, the ACL of a container object is a list of other objects, such as users, that have rights to the container, and what rights they have.

ACLs track three pieces of information: the property (what part of this object does the user have rights to?); the rights (what rights does the user have?); and the trustee (which user has these rights?). This is a very effective way to handle security. Any time a user wants to access the object, NDS checks the object to make sure that the user is listed in the ACL as a trustee of the object, and to determine what rights that user has.

When NetWare 4.1 sets up the NDS tree and base schema, a user named "Admin" is created in the Root of the tree. This user has all rights to all objects in the tree. If NetWare didn't create a user with all rights to all objects, how could other objects be created, modified, or otherwise administered? Of course, it is recommended that the first thing a network administrator does once NetWare is installed is to assign a password to the Admin user, and create other user objects with enough rights to administer separate parts of the tree.

Many of the objects in NDS are created with default ACLs attached to them. For example, when a printer is created, the creator object has Supervisor [Entry Rights] and the Supervisor [Object Rights] by default. These default ACLs are defined in NDS Object Class Definitions.

Related Topics:

Access Control Lists

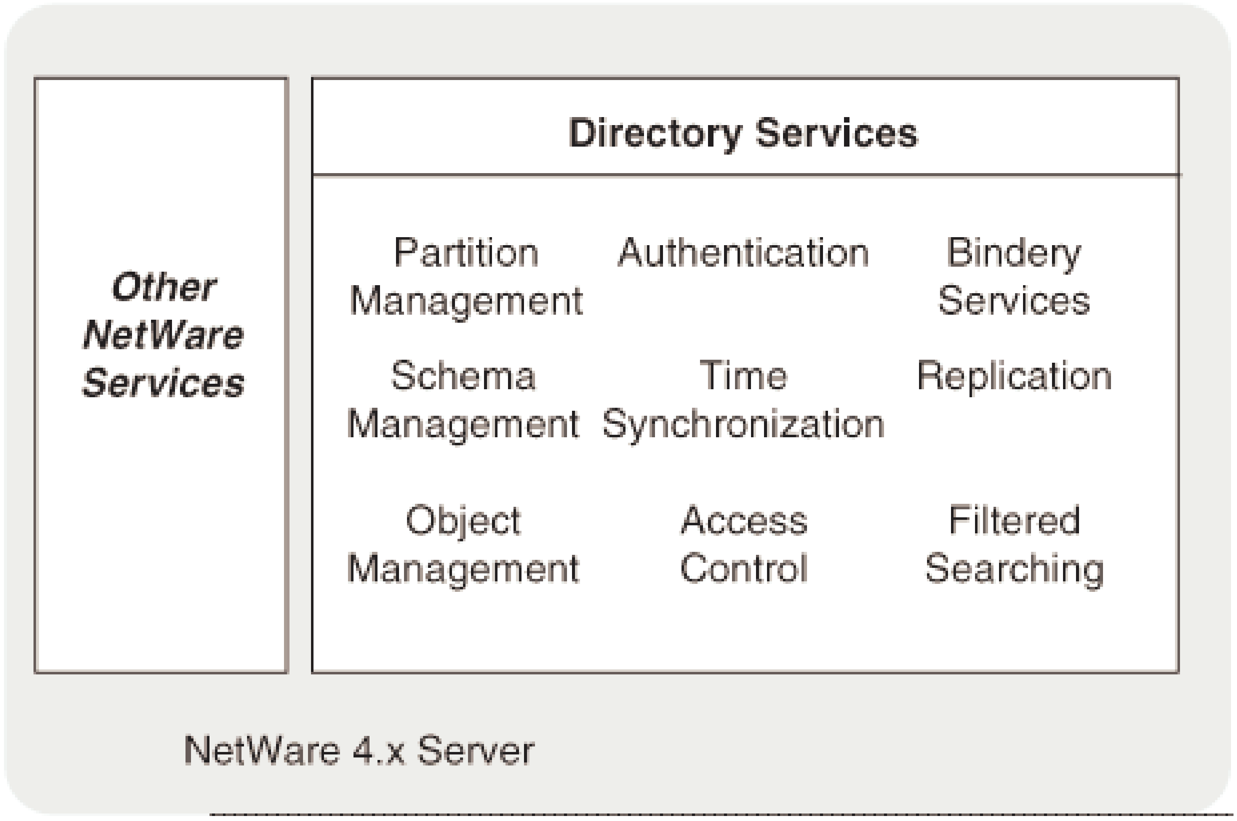
NDS Security Function

NDS Security: Concepts Guide

NDS Server Components

NDS performs many different functions in order to manage NDS requests. The following figure shows the different kinds of functions and services performed.)

Figure 17. NDS Server



Related Topics:

Table of NDS Server Utilities

NDS Agents

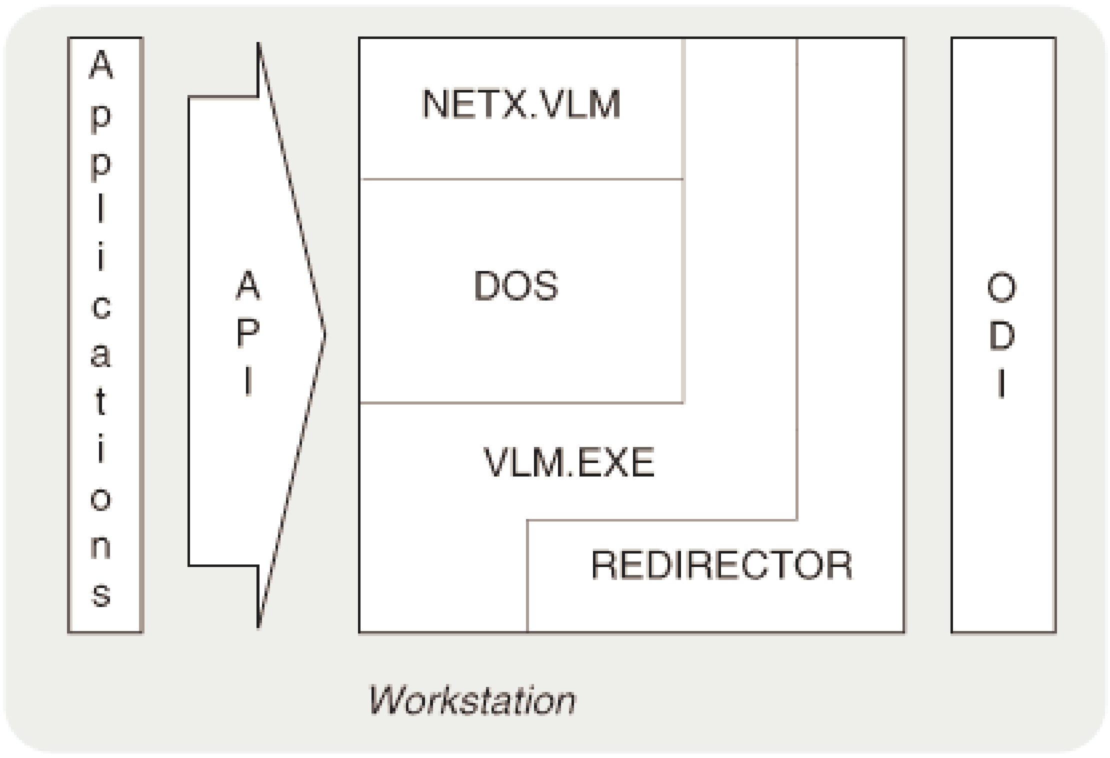
NDS Bindery

NDS Architecture: Concepts Guide

NDS Workstation Components

The following figure shows the components of a NetWare 4.x workstation. Workstation components include workstation applications and various modules for establishing and maintaining network communication.

Figure 18. NDS Workstation Architecture



The functions are linked in with the applications. They prepare and initiate requests that need to be sent to a remote server.

The NETX.VLM is Virtual Loadable Module™ (VLM™) software that handles the NETX shell API calls from applications. The NETX shell calls are the network functions used in workstations with all versions of NetWare before NetWare 4.x.

VLM.EXE is the VLM manager that handles the API calls from the application as well as managing the operation of all VLM applications. The Redirector is made up of many VLM applications that handle protocol issues and the interface with the ODI drivers.

Related Topics:

Related Topics:

Workstation Applications for NDS

Application Programming Interfaces for NDS

Shell Components for NDS

NDS Architecture: Concepts Guide

Notes About Adding NDS Objects

The function **NWDSAddObject** is used to add leaf nodes to an NDS directory. If those leaf nodes are container class objects, they can subsequently have objects attached to them. When calling this function, it is important to know the context of the object, the name you are going to give it, and have a buffer prepared containing all required data for the object. The following code segment demonstrates the use of **NWDSAddObject**:

```
NWDSCCODE N_API NWDSAddObject(
   NWDSContextHandle   context,          /* (IN) Indicates the NDS
                                                 context for the reques
   pnstr8              *objectName,      /* (IN) Points to the name of
                                                 object to be added.*/
   pnint32             *iterationHandle, /* (IN) */
   nbool8              more,             /* (IN) */
   pBuf_T              objectInfo);      /* (IN) Points to a request bu
                                                 containing the attribu
                                                 and values for the new
                                                    object. */
```

The *context* and *objectName* parameters are fairly self-explanatory. The *objectInfo* parameter points to a buffer, which is set up using **NWDSAllocBuf** and **NWDSInitBuf**.

NWDSAllocBuf is used to allocate an NDS buffer. This buffer can be of varying sizes, depending on your needs and preferences. However, Novell® has defined in NWDSDC.H two constants that are typically used with this function.

```
4096     DEFAULT_MESSAGE_LEN
64512    MAX_MESSAGE_LEN

NWDSCCODE N_API NWDSAllocBuf(
   size_t    size,  /* (IN)  Indicates the number of bytes to allocate
                             to the buffer.*/
   ppBuf_T   buf);  /* (OUT) Points to an Buf_T containing the
                               memory allocated for the buffer.*/
```

NWDSInitBuf is used to initialize an NDS buffer for an NDS request. Output buffers do not need to be initialized. For example, if you were calling **NWDSRead** and wanted all of the information for a particular object, you would not need to initialize a buffer. If, however, you wanted to request only specific information (a specific attribute), you would have to initialize a request buffer.

The following code segment demonstrates the use of **NWDSInitBuf**:

```
NWDSCCODE N_API NWDSInitBuf(
    NWDSContextHandle   context,    /*(IN) Indicates the NDS context
                                           for the request.*/
    nuint32             operation,  /*(IN) Indicates the NDS operation
                                           for which the buffer is bein
                                           initialized.*/
    pBuf_T              buf);       /*(IN) Points to the buffer being
                                             initialized.*/
```

When calling **NWDSInitBuf**, you must specify the context handle and the buffer, as indicated in the preceding code. You must also specify the intended use (allocation type) of the buffer in the operation parameter. See Buffer Allocation Types and Related Functions.

After the buffer has been created, we are ready to add the information to the buffer to create the new object. This is done by calling the **NWDSPutAttrName** and **NWDSPutAttrVal** functions. When you fill the buffer, you create a structure that logically looks something like this: attribute name --- attribute value --- attribute name --- attribute value ---.... Use alternating calls to **NWDSPutAttrName** and **NWDSPutAttrVal**. The following code segment demonstrates how you might use these functions before adding a new user.

```
ccode=NWDSPutAttrName(dContext,inBuf,"Object Class");
   /* error checking goes here */
ccode=NWDSPutAttrVal(dContext,inBuf,SYN_DIST_NAME,"User");
   /* error checking goes here */
ccode=NWDSPutAttrName(dContext,inBuf,"Surname");
   /* error checking goes here */
ccode=NWDSPutAttrVal(dContext,inBuf,SYN_CI_STRING,"Smith");
   /* error checking goes here */
```

NWDSPutAttrName is fairly straightforward and uses the following syntax:

```
NWDSCCODE N_API NWDSPutAttrName(
    NWDSContextHandle    context,   /*(IN) Indicates the NDS context
                                           for the request.*/
    pBuf_T               buf,       /*(IN) Points to the request buffer
                                           which to store the attribute
    pnstr8               attrName); /*(IN) Points to the attribute name
                                           store in the request buffer.
```

NWDSPutAttrVal is somewhat more complicated and takes the following form:

```
NWDSCCODE N_API NWDSPutAttrVal(
    NWDSContextHandle    context,  /* (IN) Indicates the NDS context
    NWDSContextHandle    context,  /* (IN) Indicates the NDS context
                                         for the request.*/
    pBuf_T               buf,      /* (IN) Points to the request buffer
                                         being prepared.*/
    nuint32              syntaxID, /* (IN) Indicates the data type of t
                                         attribute value.*/
    nptr                 attrVal); /* (IN) Points to the attribute valu
                                            be stored in the request
```

SyntaxID contains the data type of the attribute value. This data type is defined for each attribute in NDS Attribute Type Definitions. For instance, for the attribute "Object Class" above, we input the value "User" and the type SYN_DIST_NAME. You might ask, "Where do I find the attributes, values, and syntax IDs for the object I'm trying to add? Furthermore, how do I know which attributes I need?" The attribute values and syntax IDs are found in the NDS Attribute Type Definitions. If you are trying to add a User, for example, NDS Object Class Definitions defines the object User (Class).

Notice that all of the mandatory attributes as well as the optional attributes are specified. Next, for each of the attributes you want to set, go toNDS Attribute Type Definitions and NDS Attribute Syntax Definitions. "NDS Attribute Type Definitions" has an entry for each attribute type, for example the CN (Common Name) attribute.

Notice that the syntax specified for CN is Case Ignore String. NDS Attribute Syntax Definitions defines the Case Ignore String.

This gives you not only the information for the *syntaxID*, or SYN_CI_STRING, but also the form that this attribute takes; in this case, a character pointer. Some attributes take the form of more complex structures.

Now that you have the buffer set up with the appropriate attributes and values, you're ready to call **NWDSAddObject**. At this point, the function is fairly trivial. Most of the work went into setting up the buffer. As described at the beginning of this section, **NWDSAddObject** takes the following form:

```
  NWDSCCODE N_API NWDSAddObject(
      NWDSContextHandle    context,          /* (IN) Indicates the NDS
                                                   context for the requ
     pnstr8               *objectName,      /* (IN) Points to the name o
                                                   object to be added.*
     pnint32              *iterationHandle, /* (IN) */
     nbool8               more,             /* (IN) */
     pBuf_T               objectInfo);      /* (IN) Points to a request
                                                   containing the attri
                                                   and values for the n
                                                      object.*/
```

The only thing that you need to think about at this point is the name to give to the object.

Related Topics:

Change Types for NWDSPutChange

Adding an NDS Object

Adding an NDS Object: Example

NDS Access: Guides

Object Class Rules

Below is a list of rules that regulate the construction of new object classes. Clients that need to define new classes should pay close attention to these rules.

1. Object class definitions cannot be recursive. An object cannot have itself as a super class.
2. Only classes with complete structure rules can be flagged effective and used to create objects. That is, the super classes, containment, and naming attributes must be complete.
3. An effective class can be constructed three ways: a) the class defines its own structure rules, b) the class inherits structure rules from its super classes, or c) the class defines part of the structure rules (such as naming) and inherits the other part of the structure rules (such as containment) from a super class.
4. For a class that defines its own structure rules, any structure rules that might be inherited from its super classes are ignored.
5. If structure rules of an effective class are inherited, they must be nonambiguous (for more information, see "Effective and Noneffective Classes").

Related Topics:

Structure Rules

NDS Object Class Definitions

NDS Schema: Guides

Object Classes in NDS

The set of rules that controls the creation of a particular object type is called an **object class**. Object class components define the types of objects that can exist in NDS. The following types of information define an object class in accordance with the Object Class Rules:

Structure Rules

Super Class Structure

Mandatory and Optional Attributes

Related Topics:

Object Class Rules

NDS Object Class Definitions

NDS Schema: Guides

Object Management in NDS

In managing objects it is important to maintain control of changes to the ACL. 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.

In considering how to best manage objects you must therefore be very careful in giving objects the ability to change the ACL values. The privileges to be careful in granting are Write, Self, and Supervisor. 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. Say 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 could add or delete their own object as desired.You would of course maintain all privileges to the Party Group attribute yourself so you could read, add and delete any object at any time.You could in essence choose to let the individuals manage their membership in the group and/or manage it yourself.

Related Topics:

NDS Access and the File System

NDS Security: Concepts Guide

Optional Partition Attributes

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

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 the summation 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 timestamp for the last time the replica received an update.
Synchronized Up To	This attribute is a list of timestamps indicating the last time all servers holding a replica of the specified partition were synchronized.

Related Topics:

Replica Attribute

Mandatory Partition Attributes

Partition Information

Partition Class

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

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.

Related Topics:

Mandatory Partition Attributes

Partition Information

Partition Functions

These functions operate on partitions.

Function	**Comment**
NWDSAddPartition	Creates the root object of a new partition.
NWDSJoinPartitions	Joins a subordinate partition to its parent partition.
NWDSListPartitions	Lists the immediate subordinates of an object.
NWDSRemovePartition	Deletes an existing partition by deleting its master replica.
NWDSSplitPartition	Divides a partition into two partitions at a specified object.

Related Topics:

Replicas

Partitions

NDS Partition: Guides

NWDSListPartitionsExtInfo

Partition Information

Information regarding the new partition is associated with the partition's root object. This information is replicated among all the partition's replicas. The partition information includes the following:

Partition Class

Replica Attribute

Convergence Attribute

All Up To Attribute

Superior Reference

Related Topics:

Partitions

NWDSListPartitionsExtInfo

Partition Management Introduction

NDS is built on a distributed database. The database, also called the **Directory Information Base** (DIB), can be divided into multiple partitions and stored in separate locations. Each partition forms a major subtree and is a logical subset of the NDS database. Taken together, the partitions form a hierarchical tree that leads back to the root partition. Where the boundaries of two partitions meet, the one closer to the root is superior, the one farther from the root is subordinate. The tree of partitions is transparent to users.

A copy of a partition is called a **replica**. The replicas of a partition may be distributed to a number of servers.

Another section of NDS is the Schema. It is stored on every name server.

Cooperation among servers, the Requester, and NWNET library allows directory partitions to function as a single resource. NDS Partition Services let you add and delete entire partitions or individual copies (or replicas) of a partition. You can also manage the configuration of partitions by joining partitions or splitting them.

Related Topics:

Partitions

NDS Partition: Guides

Partition Replication

Partitions can be copied among multiple servers to provide fault tolerance for the NDS database. Each instance of a partition is called a 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. Nor are there restrictions regarding which partitions may reside together on a server.

You must use the master replica to create and delete subordinate partitions. You can use the master or secondary replicas to create, modify, and delete NDS information. A read-only replica can be used only to read information from the partition.

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 in the partition. The replica relies on the list of replica pointers to spread updates throughout the partition.

Related Topics:

Replica Synchronization

NDS Partition: Guides

Partition Synchronization

A partition synchronization process runs periodically on each name server and is responsible for several maintenance operations, including the synchronization of the replicas stored on that server with their counterparts on other servers.

Partition synchronization operations include:

Accepting updates from other replica synchronization processes and applying them to the relevant local replicas.

Maintaining consistency of the NDS 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 NDS replicas is directly related to the frequency with which the partition synchronization processes operate. Administrators can control the frequency of the partition synchronization to some extent according to the needs of their system by adjusting the SyncInterval.

Related Topics:

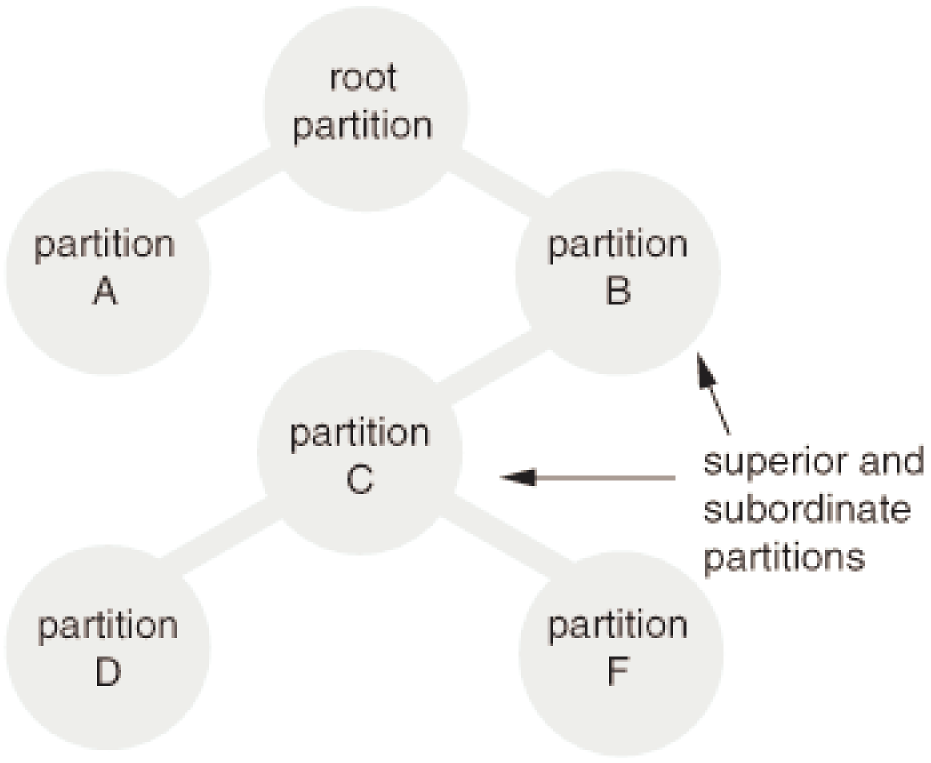
Replication Functions

NDS Partition: Guides

Partitions

Each partition forms a major subtree of NDS. Taken together, the partitions form a hierarchical tree of partitions that leads back to a **root** partition containing the root of NDS. Where the boundaries of two partitions meet, the one closer to the root is considered **superior**, and the one farther from the root is considered **subordinate**. (See the following figure.)

Figure 19. Hierarchical Partitions



It is very important for purposes of discussion to note the difference between NDS root and a partition root. NDS root is the topmost object of the entire NDS tree. It is an object named [Root].

Within each partition is a hierarchy of directory objects. Both the hierarchy of partitions and the hierarchy of objects within each partition can have as many layers as is desirable. It should be noted that the tree of partitions is transparent to the typical NDS user. Such a user sees only a global tree of NDS objects.

An NCP™ Server that is part of an NDS tree (having been installed into an NDS tree with the 4.x installation utility) is an NDS server. Multiple partitions can be stored at the same NDS server. (It is not a requirement that a server hold a replica.) There are no restrictions placed on the partitions that can be stored together at the same NDS server. For example, none of the partitions need be contiguous to each other.

Related Topics:

Subordinate Partitions

NDS Partition: Guides

Password Changing in NDS

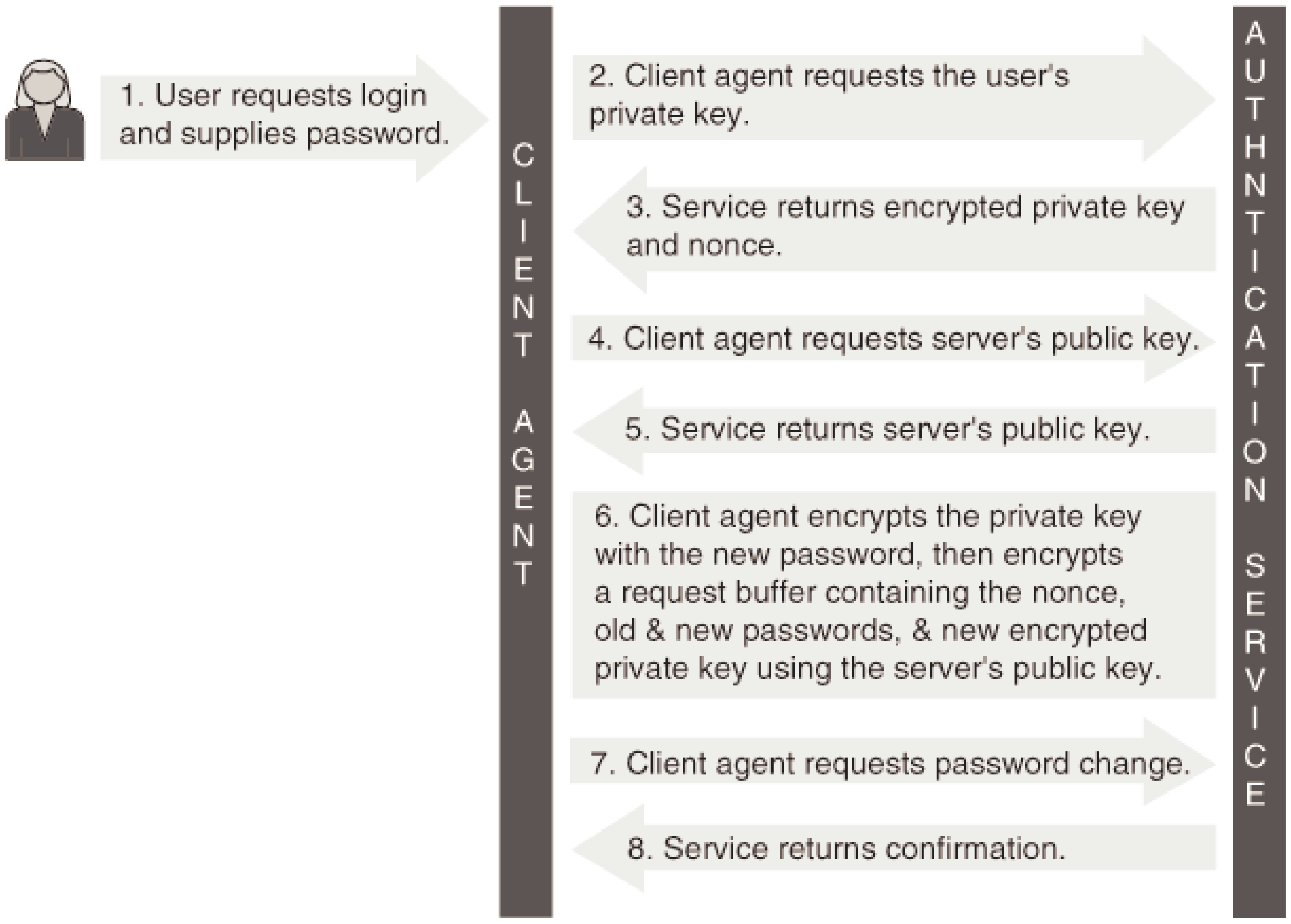
A special exchange is required in order for a client agent to change an object's password. The generation and encryption of the private key occurs at the workstation, but the encrypted key is stored with NDS. It is not stored at the user's workstation. Changing a password involves the client agent replacing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

The exchange begins with the user supplying the old and new passwords to the client agent. The client agent then asks NDS 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 NDS 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 the nonce (the old password, and the new password) are placed into a message buffer that is encrypted using the servers public key. That way this combined data can be safely transmitted to the NDS 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 20. Change Password Summary


1. User requests login and supplies password.
CLIENT AGENT
AUTHENTICATION SERVICE
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.

Related Topics:

Multiple Tree Functions

Authentication

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 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 referal 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 NDS class.

Related Topics:

Distribution of Access Control Information

NDS Partition: Guides

Public and Private Key Pairs

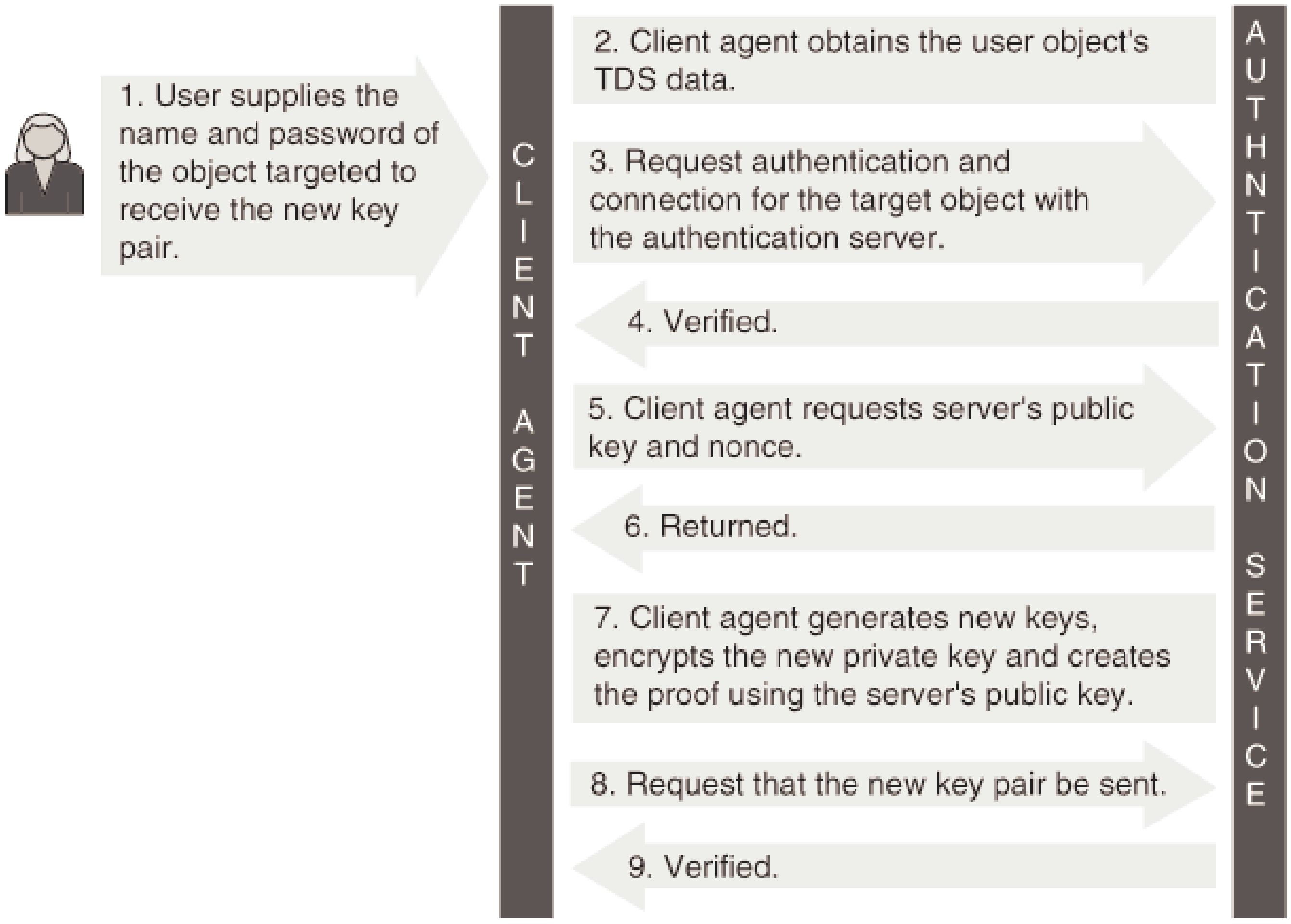
Each object in NDS has its own private/public key pair that is generated at the time the object is created. If conditions warrant it, a key pair can be changed. Setting or changing the public/private key pair for an object requires a special exchange between the client agent and Authentication. The exchange assumes that the client desiring to make the change has already authenticated itself to NDS. 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 21. Set Key Pair Summary


1. User supplies the name and password of the object targeted to receive the new key pair.
CLIENT AGENT
AUTHNTICATION SERVICE
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.

Related Topics:

Password Changing in NDS

Authentication

Read Requests

Three read function parameters (*infoType*, *allAttrs*, and *attrNames*) work together to determine what attribute information is requested.

You can specify the type of information you want returned by setting *infoType* to:

DS_ATTRIBUTE_NAMES
: NDS returns only the names of the attributes.

DS_ATTRIBUTE_VALUES
: NDS returns the names and values of the attributes.

DS_EFFECTIVE_PRIVILEGES
: NDS returns the names and the effective privileges to the attributes. The privileges returned are those of **Self** (the logged-in user).

If the *allAttrs* parameter is TRUE, NDS returns information for all attributes of the object and *attrNames* is ignored. Therefore, *attrNames* should be set the NULL. If *allAttrs* is FALSE, NDS returns the attributes named in the input buffer, which is pointed to by *attrNames*.

If *allAttrs* is FALSE and *attrNames* is NULL, no attribute information is returned. However, you can use the return value of **NWDSRead** to determine whether the object exists or access to the object is allowed.

Related Topics:

Notes About Adding NDS Objects

Reading Attributes of NDS Objects

NDS Access: Guides

Reading NDS Attribute Type Definitions

For each attribute type definition, the name of the attribute appears in large type at the first of the attribute listing followed by a brief description of the attribute's purpose. Valid abbreviations for the attribute appear in parentheses next to the attribute name. Additionally, you will find the following information:

Syntax
: The name of the syntax for this attribute type. See NDS Attribute

Syntax Definitions for the syntax specification.

Constraints

Any restrictions affecting the attribute value. The constraints, which are defined in NWDSDEFS.H, are as follows:

DS_HIDDEN_ATTR---The user cannot see or modify the attribute. A name server has created the attribute and the server maintains it.

DS_NONREMOVABLE_ATTR---The attribute cannot be removed from the object class definition.

> **NOTE:** All Base Schema Attribute Type Definitions have the nonremovable constraint.

DS_PUBLIC_READ---This constraint indicates that anyone can read the attribute. No read privileges need to be assigned. Inheritance masks cannot be used to prevent an object from reading attributes with this constraint.

DS_READ_ONLY_ATTR---The user cannot modify the attribute. A name server has created the attribute and maintains it.

DS_SERVER_READ---With this constraint on an attribute, special allowance is given to server class objects. Server class objects can read the attribute even though the privilege to read has not been inherited or explicitly granted. Inheritance masks cannot be used to restrict servers from reading attributes with this constraint.

DS_SF_PER_REPLICA---The information in this attribute is not synchronized on other replicas.

DS_SINGLE_VALUED_ATTR---The attribute has a single value only.

DS_SIZED_ATTR---The attribute has an upper and lower bound. This can be the length for strings or the value for integers.

DS_STRING_ATTR---These attributes are of string type. Attributes of this type can be used as naming attributes.

DS_SYNC_IMMEDIATE---When modifications are made to an attribute with this constraint, other replicas containing the object are synchronized immediately rather than at the next synchronization interval.

DS_WRITE_MANAGED---You must have managed rights on the object that contains this attribute before you can change the attribute's value.

Used In

The object class definitions which require or allow an attribute of this type when creating that class of object.

Remarks

These remarks may include further restrictions, how to use the attribute, references to related documents, etc.

Attributes are assigned to objects according to the object's class definition. For more information about the specific attributes an object class uses, see NDS Object Class Definitions.

Related Topics:

NDS Schema Extensions

NDS Attribute Type Definitions

Reading Class Definitions

For each object class definition, the name of the class appears in large type at the first of the listing, followed by a brief description of how the class is used. The description also specifies whether the class is effective or noneffective.

Each object class has the following information defined:

Super Classes

Objects of the class inherit information types and attributes from classes listed here. All object classes must have one or more super classes, except Top, which is a super class to all classes. The super classes are listed in a hierarchal manner, with the super class at the bottom of the list being the class that the current class inherits from (the immediate super class).

Containment

Objects of the class may only be created as subordinates in the NDS tree to objects of the classes listed here. An object of the class may not be subordinate to any object of a class that is not listed here.

Named By

The partial name or Relative Distinguished Name (RDN) of objects of the class consists of at least one of the attributes listed here. These attributes can be either mandatory or optional attributes, but at least one must be given a value when creating an object of the class. If the only Named By attribute is optional, it is in effect mandatory.

Attributes listed in this section will also be listed in the "Mandatory Attributes" and "Optional Attributes" sections. For example, a User object is named by Common Name, which is a mandatory attribute.

Mandatory and Optional Attributes.

All attributes are either mandatory or optional. If an attribute is mandatory, a value must be assigned to that attribute. If an attribute is optional, an assigned value is not required unless it is the only Named By attribute.

The Containment classes and Named By attributes of a class comprise a set of structure rules that define an object's relation to other objects in NDS. An object's RDN is determined by its Named By attributes; its distinguished name is determined by the objects it is subordinate to which must be of a class found in the Containment classes. Therefore, structure rules effectively control the formation of the distinguished names.

For example, if a User object has the common name

```
CN=Lisa
```

and her object is located in a container name

```
OU=Marketing.O=Wimple Makers.C=US
```

then the distinguished name for Lisa would be

```
CN=Lisa.OU=Marketing.O=Wimple Makers.C=US
```

Default ACL Template

Every expanded class definition has an ACL attribute (inherited from Top). This attribute holds information about which trustees have access to the object itself (entry rights) and which trustees have access to the attributes for the object. This information is stored in sets of information containing the trustee name, privileges, and the affected attribute (entry, all attributes, or a specific attribute). For example, the default template for AFP Server is that the creator of an object has supervisor privileges on [Entry Rights].

Some object classes have a default set of values for their ACL. Objects also inherit default ACL values from their super classes. Therefore, every object class inherits a default ACL template from Top. When an object is created, its ACL contains the values that are in the default ACL template for that object. There are two cases where the ACL values are different:

Your code overrides the default values.

The creator of the object has effective rights comparable to those in the default template. In this case, the rights are not granted explicitly.

Related Topics:

NDS Schema Extensions

Creating a Class Definition: Example

Reading a Class Definition: Example

NDS Object Class Definitions

Reading Syntax Definitions

For each syntax specification, the name of the syntax appears in large type at the first of the listing and is followed by a brief description of how the syntax is used.

Each syntax defines the following information:

Syntax ID

A 32-bit integer used as an identifier in Application Programming Interface (API) functions that transfer attribute values in and out of NCP™ message buffers. The syntax IDs are defined in NWDSDEFS.H.

API Data Structure

A C structure supported by the NDS™ API.

Matching Rules

The rules for matching two values that comply with the syntax.

Used In

A list of attribute type definitions that use the syntax.

Remarks

Additional information that may include information concerning comparisons, explanations of structure members, syntax examples, and so on.

To determine which syntax is used for a particular attribute type, refer to the attribute's Syntax specification in the NDS Attribute Type Definitions.

Related Topics:

NDS Schema Extensions

NDS Attribute Syntax Definitions

Registering Attribute Types and Class Definitions

When you define new attribute types or new object class definitions, you must register them with Developer Support to ensure uniqueness. To clear new types and definitions, call Developer Support at 1-800-REDWORD (1-800-733-9673). The international number is 801-429-5588. Correspondence can be sent by FAX to 801-429-2990.

Related Topics:

NDS Schema Extensions

NDS Attribute Type Definitions

NDS Schema: Guides

Relational Operators for NDS Searches

A relational operator asserts something about an attribute (for example, the attribute is present or its value is greater than 100). The truth of a relational operator is evaluated with the matching rules associated with the attribute's syntax. The following table shows the relational operators and the conditions that test TRUE for each.

Table auto. Relational Operators

Token	Value	Comment
FTOK_EQ	7	TRUE only if the attribute's value is equal to the asserted value.
FTOK_GE	8	TRUE only if the attribute's value is greater than or equal to the asserted value.
FTOK_LE	9	TRUE only if the relative ordering places the asserted value before any of the attribute's values.
FTOK_APPROX	10	TRUE only if the value of the attribute matches the asserted value. If approximate match isn't supported, this operator matches for equality.
FTOK_PRESENT	15	TRUE only if the named attribute is present in the entry.
FTOK_RDN	16	TRUE only if the object's Relative Distinguished Name matches the asserted value.
FTOK_BASECLS	17	TRUE only if the object belongs to the asserted base class.
FTOK_MODTIME	18	TRUE only if the modification timestamp is greater than or equal to the asserted value.
FTOK_VALTIME	19	TRUE only if the creation timestamp is greater than or equal to the asserted value.

You can use wildcards to create relational assertions for string values. The wildcard character is the asterisk (*). Use the backslash escape character to escape the asterisk (*) or to escape the back slash itself (\\).

Related Topics:

Logical Operators for NDS Searches

NDS Search Introduction

Replica Attribute

This attribute contains a set of replica pointers to other replicas of the partition. Each entry contains the following:

Server Name---This field is the distinguished name of the server holding the replica.

Replica Type---This field indicates whether the replica is master, secondary, or read-only.

Replica Number---This field is a unique number that identifies the server that holds a replica of this partition. This number is assigned by the server that stores the master replica.

Net Address---This field contains the last known network address of a server that holds a replica of this partition. This value is a hint and is not guaranteed, in that the address may have been changed but not yet synchronized.

Related Topics:

Convergence Attribute

Partition Information

Replica Information

The partition root object on each name server also stores information for itself that is kept locally. This information is not replicated. Replica information includes the following:

Replica State

Replica Type

Time Stamp

Related Topics:

Replicas

NDS Partition: Guides

Replica State

The *Replica State* attribute indicates the current status of the replica. The attribute can receive the following values:

0	Replica is on
1	Replica is new
2	Replica is dying
3	Partition is dying
5	Replica is dead

A replica may not be on because it is currently being installed or removed.

Related Topics:

Replica Type

Replica Information

Replica Synchronization

A vital part of NDS is the updating or **synchronization** of the NDS partitions. 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. Since updates to the replicas can occur at different times and places, at any given moment a partition is synchronized only up to a certain time. 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**.

NDS is a **loosely** synchronized database. It 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 form another replica before the update is synchronized.

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

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 Partition Synchronization process.

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

The Partition Synchronization processes each of the replicas of the partition one-by-one. After the processing for a replica has been completed, or if it fails, the Partition 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 Partition 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 Partition Synchronization process, however, proceeds with attempts to update other replicas in the replica list. The replicas that are not updated in this Partition Synchronization process are attempted again.

When the Partition Synchronization process wakes up, it builds a list of replicas for the partition it is attempting to synchronize. The Partition 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 Partition Synchronization process compares the local ReceivedUpTo attribute with the time stamp vector received. If the local ReceivedUpTo is more current than the received time stamp vector, the Partition 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 Partition 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 Partition 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 Partition 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 Partition 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.

Related Topics:

Replication Functions

NDS Partition: Guides

Replica Type

The *Replica Type* attribute indicates the type of replica according to the

The *Replica Type* attribute indicates the type of replica according to the following values:

0	Master
1	Secondary
2	Read-only

The replica type determines the client operations that can be performed on the replica.

Related Topics:

Partition Replication

Replica Information

Replicas

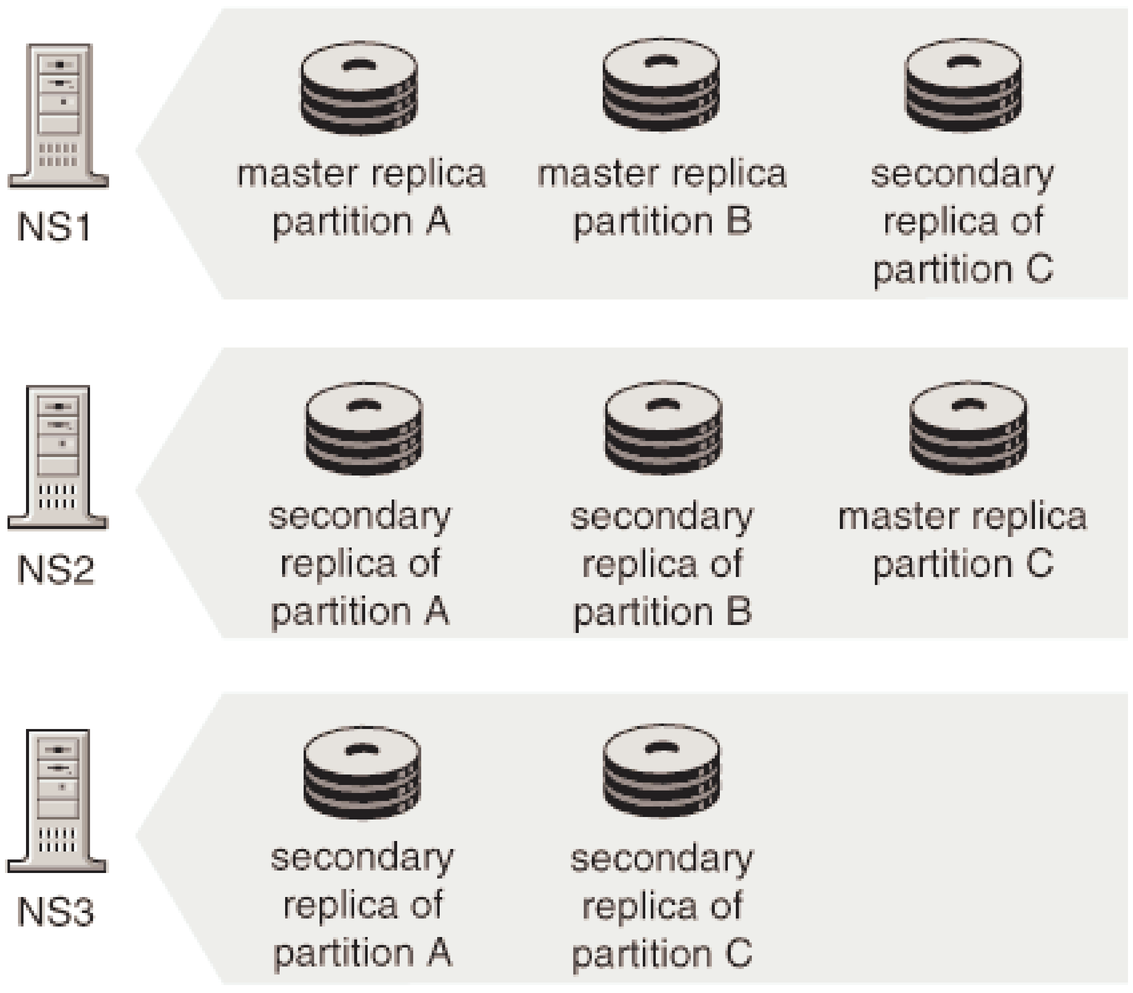
A copy of a partition is called a **replica**. Multiple replicas of a partition may exist, but only one replica of a partition may exist per server. The initial replica of a partition is called the **master** replica.

There can be only one master replica of a partition. Additional replicas are either **secondary** or **read-only** replicas; there are no restrictions on the number of these replicas.

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 functions and for all partition functions.) 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 (NS1, NS2, and NS3). NS1 stores the master replicas of partitions A and B and a read-only replica of partition C. NS2 stores the master replica of partition C and secondary replicas of A and B. NS3 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 NS1 or NS2, and NS2 or NS3 could handle such a request for partition C.

Figure 22. Replication of Partitions


NS1
master replica partition A
master replica partition B
secondary replica of partition C
NS2
secondary replica of partition A
secondary replica of partition B
master replica partition C
NS3
secondary replica of partition A
secondary replica of partition C

(The DS functions take care of sending the request to the appropriate replica. Your application does not need to worry about this.)

To create a new partition subordinate to Partition A or B the request must be directed to NS1. To create a new partition subordinate to Partition C the request must be directed to NS2.

Related Topics:

Replica Information

NDS Partition: Guides

Replication Functions

These functions operate on replicas of a partition.

Function	Comment
NWDSAddReplica	Adds a replica of an existing partition to a server.
NWDSChangeReplicaType	Changes the replica type of a given replica on a given server.
NWDSRemoveReplica	Deletes a replica from the replica set of a partition.
NWDSSyncReplicaToServer	Requests a replica to initiate synchronization with a specified server.

Related Topics:

Name Server Information

Replicas

NDS Partition: Guides

Restrictions on New Classes

A class's containment classes and naming attributes make up the class's structure rules. Structure rules determine how the class behaves in the NDS hierarchical tree. NDS places the following restrictions on how structure rules are implemented for a new class.

1. An object can't have itself as a super class.

2. If a class's structure rules are incomplete, the class is non-effective (can't be used to create objects).

3. There are three ways for a class to acquire structure rules:

   Define them itself.

   Inherit them from a super class.

Define one component (naming or containment) and inherit the other.

4. If a class defines its own structure rules, any inherited structure rules don't apply.
5. If an effective class inherits its structure rules, the structure rules must be non-ambiguous. That is, the class can't inherit rules that are contradictory or otherwise incompatible with it.

Related Topics:

NDS Class Functions

NDS Schema: Guides

Retrieval of Information from NDS

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

Related Topics:

Lookup in NDS

Browsing NDS

Searching NDS

Introduction to NDS Development: Guide

Rights to an NDS Object

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 [Object Rights]. The rights a user (or any other object) might have to another object include Browse, Create, Delete, Rename, and Supervisor. Let's take a look at these rights in a little more detail.

Browse

This right lets the user "see" an object. If you were getting a list of all objects that were available, and you didn't have Browse rights to a particular object, say 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.

Rename

You must have Rename rights in order 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.

Related Topics:

Access Privileges

Rights to the Properties of an NDS Object

NDS Security: Concepts Guide

Rights to the Properties of an NDS Object

Now that we have determined the rights available for gaining access to an object, how do we actually getting the data associated with that object? This is where property rights come in.

A user (remember this could be any 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 a user needs to be able to see the names, addresses, and phone numbers of other users, we want to make sure the user has rights to each of those properties of the other users. We might decide to give the user rights to all of the attributes (properties) of an object, or to only those attributes the user needs to access. The syntax used to denote rights to all properties of an object is [All Property Rights]. If rights are given to a specific property, the syntax used is merely the name of that property. Whether we are talking about [All Property Rights] or rights to specific properties, the rights available are the same.

Descriptions of each of the property rights of an object follows:

Compare

Compare rights gives our 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 the user the ability to see the values of the properties that are not hidden properties. By having Read rights, the user automatically gets Compare rights, as implicit (implied) rights.

Add or Delete Self

With this right, the user can add its own object as the value of the property. For example, if the user has Add or Delete Self rights to the Member property of an object, such as a container, 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 a property, as long as that property is not a Read Only property. Having Write rights implies Add or Delete Self rights.

Supervisor

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

To review, an object can have the following rights to access other objects: Browse, Create, Delete, Rename, and Supervisor. And an object can have the following rights to the properties of other objects: Compare, Read, Add or Delete Self, and Supervisor. This list of rights is very comprehensive and efficient, which "covers all the bases" in administering NDS security.

Related Topics:

Giving Rights in NDS

Rights to an NDS Object

NDS Security: Concepts Guide

Schema Changes for NetWare 4.01

With the NetWare 4.01 version of the NetWare OS, the NDS™ Base Schema received minor changes. The differences between the NetWare 4.0 Base Schema and the NetWare 4.01 Base Schema are listed in this section.

NDS Object Class Definitions

The changes to the individual object class definitions are shown in the following table.

Table AUTO. Changes to Individual Class Definitions

Object Class	**Change**

Partition	Added *Partition Control* as a optional attribute.
Top	Added *Revision* as an optional attribute.

Because of inheritance, the changes listed in the previous table flow to other classes. These changes, which show up in the expanded class definitions, are listed in the following table.

Table AUTO. Changes to Expanded Class Definitions

Object Class	**Change**
All Classes	Added *Revision* as an optional attribute as a result of inheritance from Top.

Attribute Definitions

The changes made to the attribute definitions are shown in the following table.

Table AUTO. Changes to Attribute Definitions

Attribute	**Change**
Group Membership	Added the DS_WRITE_MANAGED constraint.
Higher Privileges	Added the DS_WRITE_MANAGED constraint.
Partition Control	Created as a new attribute definition.
Passwords Used	Removed the DS_SINGLE_VALUED_ATTR constraint.
Revision	Created as a new attribute definition.
Security Equals	Added the DS_WRITE_MANAGED constraint.
Synchronized Up To	Added the DS_SF_PER_REPLICA constraint.

Syntax Definitions

The changes made to the syntaxes for the NetWare 4.01 Base Schema are shown in the following table.

Table AUTO. Changes to Syntax Definitions

Syntax	**Change**
Counter	Added its use by *Revision*

Typed Name	Added its use by *Partition Control*

Related Topics:

Schema Changes for NetWare 4.1

NDS Schema: Guides

Schema Changes for NetWare 4.1

The NDS Base Schema for the NetWare 4.1™ OS has received a number of extensions to the Base Schema of the 4.01 OS. These changes are listed in this section.

The following object-class definitions have been added to the 4.1 Base Schema:

Table AUTO. New NDS Object Class Definitions

Object Class	**Change**
External Entity	New class definition
List	New class definition
Message Routing Group	New class definition
Messaging Server	New class definition

The following changes have been made to the existing individual object-class definitions for the 4.1 Base Schema:

Table AUTO. Changes to Individual Class Definitions

Object Class	**Change**
Group	Added EMail Address, Login Script, Mailbox ID, Mailbox Location, Profile, and Profile Membership as optional attributes.
NCP Server	Added DS Revision and Messaging Server as optional attributes.
Organization	Added Mailbox ID and Mailbox Location as optional attributes.
Organizational Person	Added Mailbox ID and Mailbox Location as optional attributes.
Organizational Role	Added Mailbox ID and Mailbox Location as optional attributes.

Organizational Unit	Added Mailbox ID and Mailbox Location as optional attributes.
Partition	Added Replica Up To as an optional attribute.
Person	Added Generational Qualifier and Initials as optional attributes.
Profile	Added Full Name as an optional attribute.
Server	Added Security Equals and Security Flags as optional attributes.
Top	Added Authority Revocation, CA Private Key, CA Public Key, Certificate Revocation, Certificate Validity Interval, Cross Certificate Pair, Equivalent To Me, and Last Referenced Time as optional attributes.
User	Added Profile Membership and Security Flags as optional attributes.

Because of inheritance, the above changes to individual class definitions flow to some of the other classes. The following table shows how the changes listed above affect the expanded class definitions of other classes. (This listing shows only the effects through inheritance.)

Table AUTO. Changes to Expanded Class Definitions

Object Class	**Change**
All Classes	Inherit Authority Revocation, CA Private Key, Certificate Revocation, Certificate Validity Interval, Cross Certificate Pair, Equivalent To Me, and Last Referenced Time as optional attributes from Top
AFP Server	Inherits Security Equals and Security Flags as optional attributes from Server.
NCP Server	Inherits Security Equals and Security Flags as optional attributes from Server.
Print Server	Inherits Security Equals and Security Flags as optional attributes from Server.

The following attributes have been added to the 4.1 Base Schema

Table AUTO. New Attributes

Attribute	**Used In**
Certificate Validity Interval	Top
DS Revision	NCP Server

Equivalent To Me	Top
External Name	External Entity
External Synchronizer	None
Generational Qualifier	Person
Given Name	Person
Initials	Person
Last Referenced Time	Top
Mailbox ID	Group, List, Organization, Organizational Person, Organizational Role, and Organizational Unit
Mailbox Location	Group, List, Organization, Organizational Person, Organizational Role, and Organizational Unit
Message Routing Group	Messaging Server
Messaging Database Location	Messaging Server
Messaging Server	NCP Server
Messaging Server Type	Messaging Server
Postmaster	Messaging Server
Profile Membership	Group and User
Replica Up To	Partition
Security Flags	Server and User
Supported Gateway	Messaging Server

The following changes have been made to the attributes for the 4.1 Base Schema.

Table AUTO. Changes to Attributes

Attribute	Change
Account Balance	Added DS_SYNC_IMMEDIATE constraint.
ACL	Added DS_SYNC_IMMEDIATE constraint.
Aliased Object Name	Added DS_SYNC_IMMEDIATE constraint.
Allow Unlimited Credit	Added DS_SYNC_IMMEDIATE constraint.
Authority Revocation	Added use by Top. Added DS_SYNC_IMMEDIATE constraint.
C	Added DS_SYNC_IMMEDIATE constraint.
CA Private Key	Added use by Top.

CA Private Key	Added use by Top.
CA Public Key	Added use by Top.
Cartridge	Added DS_SYNC_IMMEDIATE constraint.
Certificate Revocation	Added use by Top. Added DS_SYNC_IMMEDIATE constraint.
CN	Added use by External Entity and List. Added DS_SYNC_IMMEDIATE constraint.
Convergence	Added DS_SYNC_IMMEDIATE constraint.
Cross Certificate Pair	Added use by Top. Added DS_SYNC_IMMEDIATE constraint.
Default Queue	Added DS_SYNC_IMMEDIATE constraint.
Description	Added use by External Entity and List. Added DS_SYNC_IMMEDIATE constraint.
Detect Intruder	Added DS_SYNC_IMMEDIATE constraint.
Device	Added DS_SYNC_IMMEDIATE constraint.
EMail Address	Added use by External Entity, Group, and List. Added DS_PUBLIC_READ constraint. Added DS_SYNC_IMMEDIATE constraint.
Fascimile Telephone Number	Added use by External Entity. Added DS_SYNC_IMMEDIATE constraint.
Full Name	Added use by Profile. Added DS_SYNC_IMMEDIATE constraint.
GID	Added DS_SYNC_IMMEDIATE constraint.
Group Membership	Added use by External Entity.
High Convergence Sync Interval	Added DS_SYNC_IMMEDIATE constraint.
Home Directory	Added DS_SYNC_IMMEDIATE constraint.
Host Device	Added DS_SYNC_IMMEDIATE constraint.
Host Resource Name	Added DS_SYNC_IMMEDIATE constraint.
Host Server	Added DS_SYNC_IMMEDIATE constraint.
Inherited ACL	Added DS_SYNC_IMMEDIATE constraint.
Intruder Attempt Reset Interval	Added DS_SYNC_IMMEDIATE constraint.
Intruder Lockout Reset Interval	Added DS_SYNC_IMMEDIATE constraint.
L	Added use by External Entity and List. Added DS_SYNC_IMMEDIATE constraint.
Language	Added DS_SYNC_IMMEDIATE constraint.
Locked by Intruder	Added DS_SYNC_IMMEDIATE constraint.

Lockout After Detection	Added DS_SYNC_IMMEDIATE constraint.
Login Allowed Time Map	Added DS_SYNC_IMMEDIATE constraint.
Login Disabled	Added DS_SYNC_IMMEDIATE constraint.
Login Expiration Time	Added DS_SYNC_IMMEDIATE constraint.
Login Grace Limit	Added DS_SYNC_IMMEDIATE constraint.
Login Grace Remaining	Added DS_SYNC_IMMEDIATE constraint.
Login Intruder Address	Added DS_SYNC_IMMEDIATE constraint.
Login Intruder Attempts	Added DS_SYNC_IMMEDIATE constraint.
Login Intruder Limit	Added DS_SYNC_IMMEDIATE constraint.
Login Intruder Reset Time	Added DS_SYNC_IMMEDIATE constraint.
Login Maximum Simultaneous	Added DS_SYNC_IMMEDIATE constraint.
Login Script	Added use by Group. Added DS_SYNC_IMMEDIATE constraint.
Low Convergence Reset Time	Added DS_SYNC_IMMEDIATE constraint.
Low Convergence Sync Interval	Added DS_SYNC_IMMEDIATE constraint.
Member	Added use by List. Added DS_SYNC_IMMEDIATE constraint.
Memory	Added DS_SYNC_IMMEDIATE constraint.
Message Server	Added DS_SYNC_IMMEDIATE constraint.
Minimum Account Balance	Added DS_SYNC_IMMEDIATE constraint.
Network Address	Added DS_SYNC_IMMEDIATE constraint.
Network Address Restriction	Added DS_SYNC_IMMEDIATE constraint.
NNS Domain	Added DS_SYNC_IMMEDIATE constraint.
Notify	Added DS_SYNC_IMMEDIATE constraint.
O	Added use by List. Added DS_SYNC_IMMEDIATE constraint.
Obituary	Added DS_SYNC_IMMEDIATE constraint.
Object Class	Added DS_SYNC_IMMEDIATE constraint.
Operator	Added DS_SYNC_IMMEDIATE constraint.

OU	Added use by External Entity and List. Added DS_SYNC_IMMEDIATE constraint.
Owner	Added use by List. Added DS_SYNC_IMMEDIATE constraint.
Page Description Language	Added DS_SYNC_IMMEDIATE constraint.
Partition Control	Added DS_SYNC_IMMEDIATE constraint.
Partition Creation Time	Added DS_SYNC_IMMEDIATE constraint.
Password Allow Change	Added DS_SYNC_IMMEDIATE constraint.
Password Expiration Interval	Added DS_SYNC_IMMEDIATE constraint.
Password Expiration Time	Added DS_SYNC_IMMEDIATE constraint.
Password Minimum Length	Added DS_SYNC_IMMEDIATE constraint.
Password Required	Added DS_SYNC_IMMEDIATE constraint.
Password Unique Required	Added DS_SYNC_IMMEDIATE constraint.
Passwords Used	Added DS_SYNC_IMMEDIATE constraint.
Path	Removed DS_SINGLE_VALUED_ATTR constraint. Added DS_SYNC_IMMEDIATE constraint.
Physical Delivery Office Name	Added use by External Entity. Added DS_SYNC_IMMEDIATE constraint.
Postal Address	Added use by External Entity. Added DS_SYNC_IMMEDIATE constraint.
Postal Code	Added use by External Entity. Added DS_SYNC_IMMEDIATE constraint.
Postal Office Box	Added use by External Entity. Added DS_SYNC_IMMEDIATE constraint.
Print Job Configuration	Added DS_SYNC_IMMEDIATE constraint.
Print Server	Added DS_SYNC_IMMEDIATE constraint.
Printer	Added DS_SYNC_IMMEDIATE constraint.
Printer Configuration	Added DS_SYNC_IMMEDIATE constraint.
Printer Control	Added DS_SYNC_IMMEDIATE constraint.
Profile	Added DS_SYNC_IMMEDIATE constraint.
Queue	Added DS_SYNC_IMMEDIATE constraint.
Queue Directory	Added DS_SYNC_IMMEDIATE constraint.

Received Up To	Added DS_SYNC_IMMEDIATE constraint.
Replica	Added DS_SYNC_IMMEDIATE constraint.
Resource	Added DS_SYNC_IMMEDIATE constraint.
Revision	Added DS_SYNC_IMMEDIATE constraint.
Role Occupant	Added DS_SYNC_IMMEDIATE constraint.
S	Added use by External Entity. Added DS_SYNC_IMMEDIATE constraint.
SA	Added use by External Entity. Added DS_SYNC_IMMEDIATE constraint.
SAP Name	Added DS_SYNC_IMMEDIATE constraint.
Security Equals	Added use by Server.
See Also	Added use by External Entity and Group. Added DS_SYNC_IMMEDIATE constraint.
Serial Number	Added DS_SYNC_IMMEDIATE constraint.
Server	Added DS_SYNC_IMMEDIATE constraint.
Server Holds	Added DS_SYNC_IMMEDIATE constraint.
Status	Added DS_PUBLIC_READ constraint. Added DS_SYNC_IMMEDIATE constraint.
Supported Connections	Added DS_SYNC_IMMEDIATE constraint.
Supported Services	Added use by Messaging Server. Added DS_SYNC_IMMEDIATE constraint.
Supported Typefaces	Added DS_SYNC_IMMEDIATE constraint.
Surname	Added DS_PUBLIC_READ constraint. Added DS_SINGLE_VALUED_ATTR constraint. Added DS_SYNC_IMMEDIATE constraint.
Telephone Number	Added DS_SYNC_IMMEDIATE constraint.
Title	Added use by External Entity. Added DS_SYNC_IMMEDIATE constraint.
Type Creator Map	Added DS_SYNC_IMMEDIATE constraint.
UID	Added DS_SYNC_IMMEDIATE constraint.
User	Added DS_SYNC_IMMEDIATE constraint.
Version	Added DS_PUBLIC_READ constraint. Added DS_SYNC_IMMEDIATE constraint.
Volume	Added DS_SYNC_IMMEDIATE constraint.

The following changes have been made to the syntax definitions for the 4.1 Base Schema:

Table AUTO. Changes to Syntax

Syntax	**Change**
Case Ignore String	Added use by Generational Qualifier, Given Name, Initials, Mailbox ID, Messaging Server Type, and Supported Gateway.
Distinguished Name	Added use by Equivalent To Me, Mailbox Location, Message Routing Group, Messaging Server, Postmaster, and Profile Membership.
Integer	Added use by DS Revision and Security Flags.
Interval	Added use by Certificate Validity Interval.
Octet String	Added use by External Name, External Synchronizer, and Replica Up To.
Path	Added use by Messaging Database Location.
Timestamp	Added use by Last Referenced Time.

Related Topics:

Schema Changes for NetWare 4.01

NDS Schema: Guides

Search Expression Trees

The nodes in a search filter combine to form an expression tree. To build the tree, add each node by calling **NWDSAddFilterToken**.

A node requires a token, a value, and a syntax. The token identifies either a relational operator, a logical operator, a left or right parenthesis, an attribute name (FTOK_ANAME) or an attribute value (FTOK_AVAL), or the last node of the expression (FTOK_END).

The value applies only to attribute names and attribute values. The syntax applies only to an attribute values. If the token is an operator or parenthesis, you pass a null for the value and syntax. If the token is an attribute name, you pass a null for the syntax. You must add an ending token (FTOK_END) to mark the end of an expression.

As an example, suppose you want to search for all User objects whose surname begins with "Sm." If you were expressing this criteria as a string, it would look like this:

```
Object Class=User AND Surname=Sm*
```

To create the expression tree, think of each token as corresponding to a node in the tree. The sequence for adding the tokens is the same as if you were processing the string from left to right. You add one token each for*Object Class*, = ,*User*, AND, *Surname*, = , and *Sm**.

Related Topics:

Expression Filters for NDS Searches

NDS Search Introduction

Search Filters

You search for information in NDS by building a search filter and putting the filter into the input buffer. Search filters are used only in searching. The search filter describes the set of conditions that satisfy the search. To allocate a search filter, call the **NWDSAllocFilter** function which allocates space for the filter. Call the **NWDSAddFilterToken** function to add search criteria to the allocated filter.

The finished search filter must be placed into an input buffer by calling the **NWDSPutFilter** function which normally frees the space that has been allocated to the filter. If the **NWDSPutFilter** function is not called or does not succeed, you should free the filter yourself by calling the **NWDSFreeFilter** function.

A search filter is made up of filter nodes. Each node in the tree is either a filter token or a null value. A token can be any of the following:

Attribute name

Attribute value

Relational operator

Logical operator

A token can also parenthesize groups of tokens.

When you allocate a search filter by calling the **NWDSAllocFilter** function, you receive a filter cursor. NWFILTER_CURSOR contains the filter cursor data. The cursor is used as input to other functions that operate on search filters.

Related Topics:

Relational Operators for NDS Searches

NDS Search Introduction

Search Parameters

The **NWDSSearch** function searches for NDS information and requires an input buffer to hold the search criteria. If you want to specify the attributes to be returned, you must prepare a second input buffer.

If you set the *searchAliases* parameter to TRUE, the operation will dereference any aliases encountered. (The DCV_DEREF_ALIASES flag in the directory context must also be TRUE.)

Related Topics:

Search Filters

Searching NDS

NDS Search Introduction

Searching NDS

Searching is one of the more powerful searching tools provided by NDS. This allows the user to search for specific categories of information. The user specifies the criteria for the category, and NDS 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 NDS API.

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

Related Topics:

Security of Information in NDS

Retrieval of Information from NDS

Security of Information in NDS

As an information service, NDS has an obvious requirement to maintain a secure environment. The problems with NDS security are twofold, defined respectively as **authentication** and **access control**. The two security problems are related in that effective access control depends on successful authentication. Both services can be applied to network situations outside the operation of NDS.

Related Topics:

Security through NDS Authentication

Security through Access Control

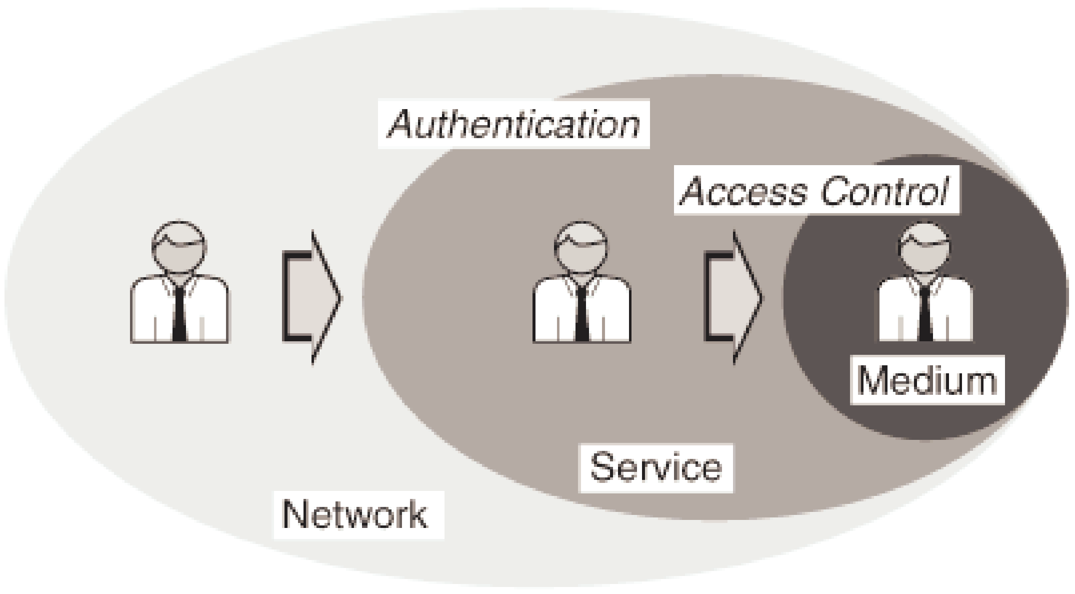
Introduction to NDS Development: Guide

Security through Access Control

NDS Authentication Service is able to confirm a user's identity and vouchsafe that identity to other services. One such other service is provided by NDS itself. NDS controls access to NDS information, according to pre-established limits. The access limitations include whether a user can create, read, or modify a piece of information.

The following figure shows the relationship between authentication and access control. In this figure, authentication is a service provided by NDS, and access control could be provided by any service.

Figure 23. NDS Security Model



The access control in the figure might be to control access to NDS information. Access control might also be to control access to a NetWare file system or to a printer or any other medium that demands access control. Different services provide different access controls. Once you have been authenticated, you then have access to the service that controls your access to the medium.

A user that has not been authenticated might nevertheless have some access to the network. For example, such a user might be able to receive unauthenticated services or carry on peer-to-peer dialogues. NDS confers **public** status on unauthenticated users. The amount of information such users can receive from NDS is determined by directory administrators.

A user might want to open a NetWare file system file. Access to that file is controlled by NetWare File System Services, not NDS. To gain access to a file on a server, the user must be authenticated to the server and must obtain a connection from the server's available connections. Even then, the file system access mechanism on the server can restrict the authenticated user from accessing the desired file.

Related Topics:

Administration of Information in NDS

Security of Information in NDS

Security through NDS Authentication

Authentication is concerned with the identities of objects. When a user establishes a session with NDS, all subsequent interactions between NDS and the user are predicated on the user's identity. NDS needs to know with some certainty that the user is authentic and is not being impersonated. Since there is no inherent method for recognizing an object across the network, NDS uses an authentication process that allows the object to validate its identity.

Authentication Services use public/private key cryptography to issue authentication materials to network clients. A client uses these materials to create an authentication mechanism for **signing** messages. The message signature allows a recipient of a message to verify the message's origin.

In establishing an initial connection to a server, a client is first required to procure authentication materials from the portion of NDS called Authentication Services. Once these materials are obtained, a signature is generated and used to sign a **proof**. The NDS checks the signature of the proof to validate it. If the proof is valid, the connection is established. The same authentication materials can be used to authenticate connections to multiple servers.

At the client workstation, authentication is performed by the NDS functions. Initially, a user must produce a password to unwrap the authentication materials which are used to create the signature and proofs. Subsequently, authentication can occur in the background using the same authentication materials. It is important to note that authentications that follow initial authentication occur transparently to the network user.

Related Topics:

Security through Access Control

Security of Information in NDS

Shell Components for NDS

Full compatibility with all versions of NetWare applications requires use of new workstation software. Since the VLM manager now handles the network requests, a new way to handle the NETX shell function calls must be used. The NETX.VLM is the VLM that provides for handling of all the old shell functions.

A NetWare 4.x workstation consists of a number of VLM applications that each serve a function and purpose. Certain VLMs are required in order to provide the basic network interface capabilities. Aside from the required VLM applications, you can further enable your workstation by loading other VLMs that are meant to provide specific capabilities. The DOS Requester VLM controls the flow of requests to the network as requested by the application. A NetWare 4.x workstation can function without NETX.VLM, but without it any application that makes a shell call does not work.

Related Topics:

NDS Server Components

NDS Workstation Components

Splitting and Joining Partitions

NDS allows a partition to be split at a designated object. Subsequent relocation of the replicas can be done through other NDS management functions.

NDS allows a subordinate and its superior partition to be merged into one, at a specified object.

Related Topics:

Partition Synchronization

NDS Partition: Guides

Structure of Output Buffers

Output buffers receive a sequence of Object_Info_T structures containing information about objects subordinate to the specified object. The Object_Info_T structure is defined with the following fields:

```
typedef struct
{
   nuint32   objectFlags;                          // Defines constra
   nuint32   subordinateCount;                     // Number of subor
   time_t    modificationTime;                     // Time last modif
   char      baseClass[MAX_SCHEMA_NAME_BYTES + 2];  // Name of Base cl
} Object_Info_T;
```

Related Topics:

NDS Buffer Allocation and Initialization Functions

Preparing NDS Output Buffers

NDS Access: Guides

Structure Rules

The Structure Rules define the relationships between objects in NDS. These relationships are defined by two properties: Containment Classes and Name Rules. Containment classes determine where the object may appear in the NDS tree relative to other classes of objects. Name rules---sometimes called **named by** attributes---determine how the object is named.

Related Topics:

Super Class Structure

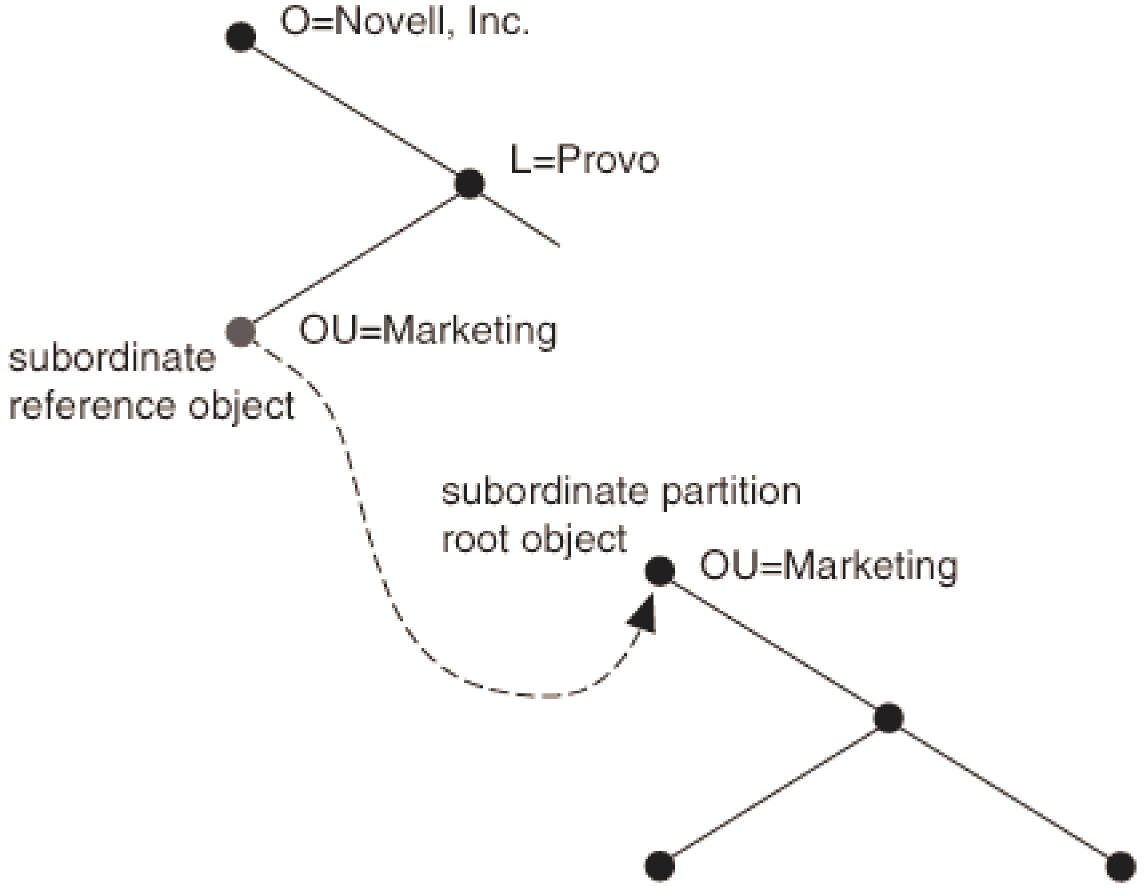
Contained By Classes

NDS Schema: Guides

Subordinate Partitions

When proceeding down the NDS tree, partitions are located by subordinate references. A subordinate reference is in the form of an object in the superior partition that refers to the subordinate partition. The subordinate reference object has a Replica attribute that lists the name servers where replicas of the subordinate are stored. It also contains information about the type of replicas there are, how many there are, and what the inherited access privileges for the partition are.

Figure 24. Subordinate References


O=Novell, Inc.
L=Provo
OU=Marketing
subordinate reference object
subordinate partition root object
OU=Marketing

Related Topics:

Partition Information

NDS Partition: Guides

Super Class Structure

Super classes provide structure for the Schema itself, not the NDS tree. The association of one object class to another is established through super class designations.

The complete definition of each object class is derived from the components of the object class plus the components of all classes found in its super class lineage. The only class that has no super class is **Top**, which is the super class of all other classes. An object class inherits the characteristics of its super classes.

Each new object class must have a **Super Classes** list. The super classes denote the Schema inheritance rules. To derive the complete set of rules for a given object class, you must look at that class' super classes. An object class inherits all the features of its super classes. Hierarchies of classes develop through class inheritance in this manner. The classes at the top of a hierarchy provide general characteristics, while those at the bottom become more and more specialized. The complete set of rules for an object class is called the **expanded class definition**.

At the top of the hierarchy for every class is the Top class. Top has no super classes. Top is used to specify information that pertains to all classes. For example, Top specifies an *Object Class* attribute and an *Access Control* attribute.

The class from which an object is created is called the object's **base class**. The information associated with an object includes the base class and the sum of the information specified by all its super classes. For purposes of searching NDS, an object is considered a member of all of its super classes. For example, the base class for User is Organizational Person. User also inherits information from the Person and Top classes. (See NDS Object Class Definitions.)

You should note that the hierarchy of object classes is distinct from the hierarchy of object names in the NDS tree. Although the Schema is stored with the rest of NDS, Schema data is logically separated from NDS and must be accessed through a different functions. Also, the Schema's class hierarchy does not form a simple tree graph, since a class can be derived from two or more classes that are superior to it. Using more than one class as a super class is called **multiple inheritance**.

Related Topics:

Inheritance

Class Inheritance Rules

Effective and Noneffective Classes

NDS Schema: Guides

Table of NDS Server Utilities

The utilities listed in the following table run at the server.

Table auto. Server Utilities

Utility	Function
Install	Install operations create the initial NDS directory structure and the network's initial NDS server.
Repair	Repair operations assess the integrity of data in NDS and perform database surgery when necessary.

Related Topics:

NDS Agents

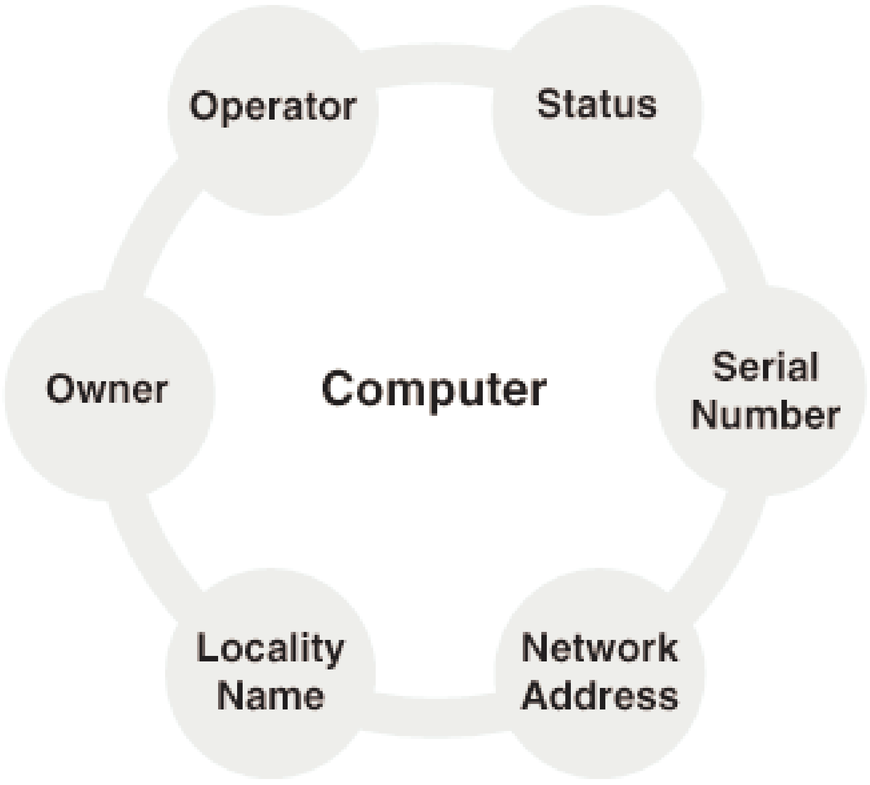
NDS Server Components

Types of Information Stored in NDS

The services provided by NDS are information services. Users (humans and computer programs) use NDS to obtain information. The information stored in NDS can be read as well as modified by users who have been granted the applicable access rights.

The information stored in NDS 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 properties (such as **Operator**, **Status**, etc.). The values of the properties are not shown in the figure, but an example of a value for **Owner** might be "Fred" which would be the name of another object in NDS.

Figure 25. An Object as a Set of Attributes



The objects in NDS 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 NDS. Specific objects might be US, Novell, Fred, FS1, PrintServer, etc. The objects are intended to fit into a human being's conception of things, not necessarily a computer's.

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

Related Topics:

Names Stored in NDS

Addresses Stored in NDS

Lists Stored in NDS

Descriptions Stored in NDS

Introduction to NDS Development: Guide

Tree Walking

When a client agent submits a request to NDS, the request is not always received by a name server that is qualified to fulfill the request. The name server receiving the request is faced with the problem of finding 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.

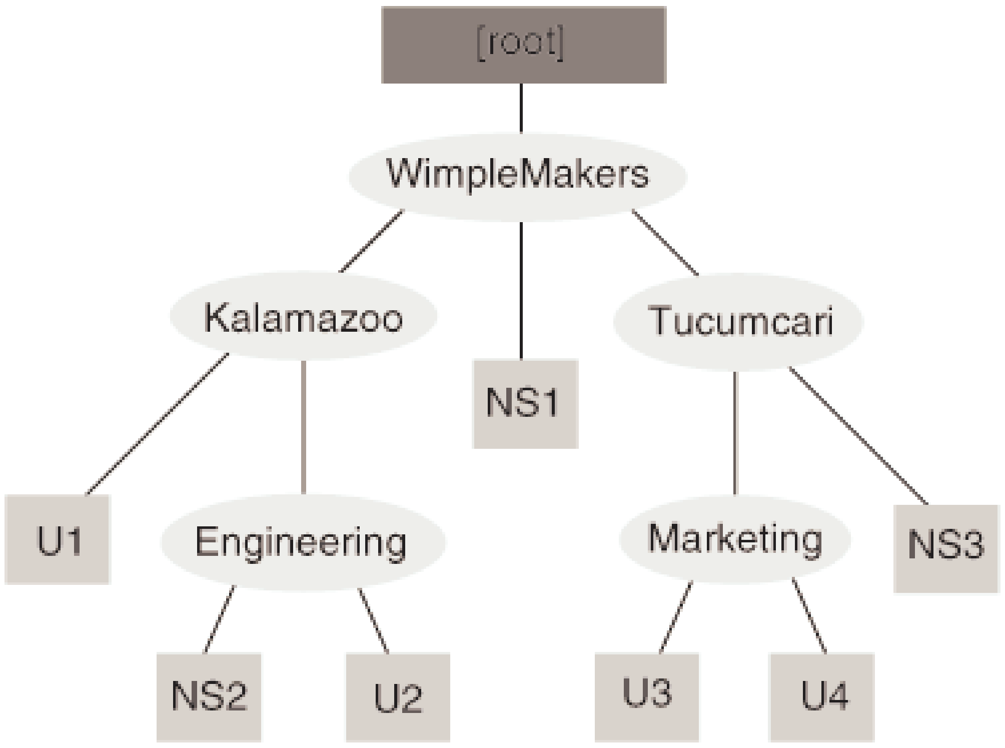
All NDS requests can be broken down to one or more names identifying objects for which information is needed. In pursuing each name component in a request, the name server searches for a partition that contains some part of the name path associated with the request. Once a partition is found, the search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

The tree walking process relies on subordinate references to connect the tree. 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 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 figure 17 shows an example NDS tree. This is the total tree structure and shows no partition designations and does not give any indication as to which servers the partitions and/or replicas are stored on.

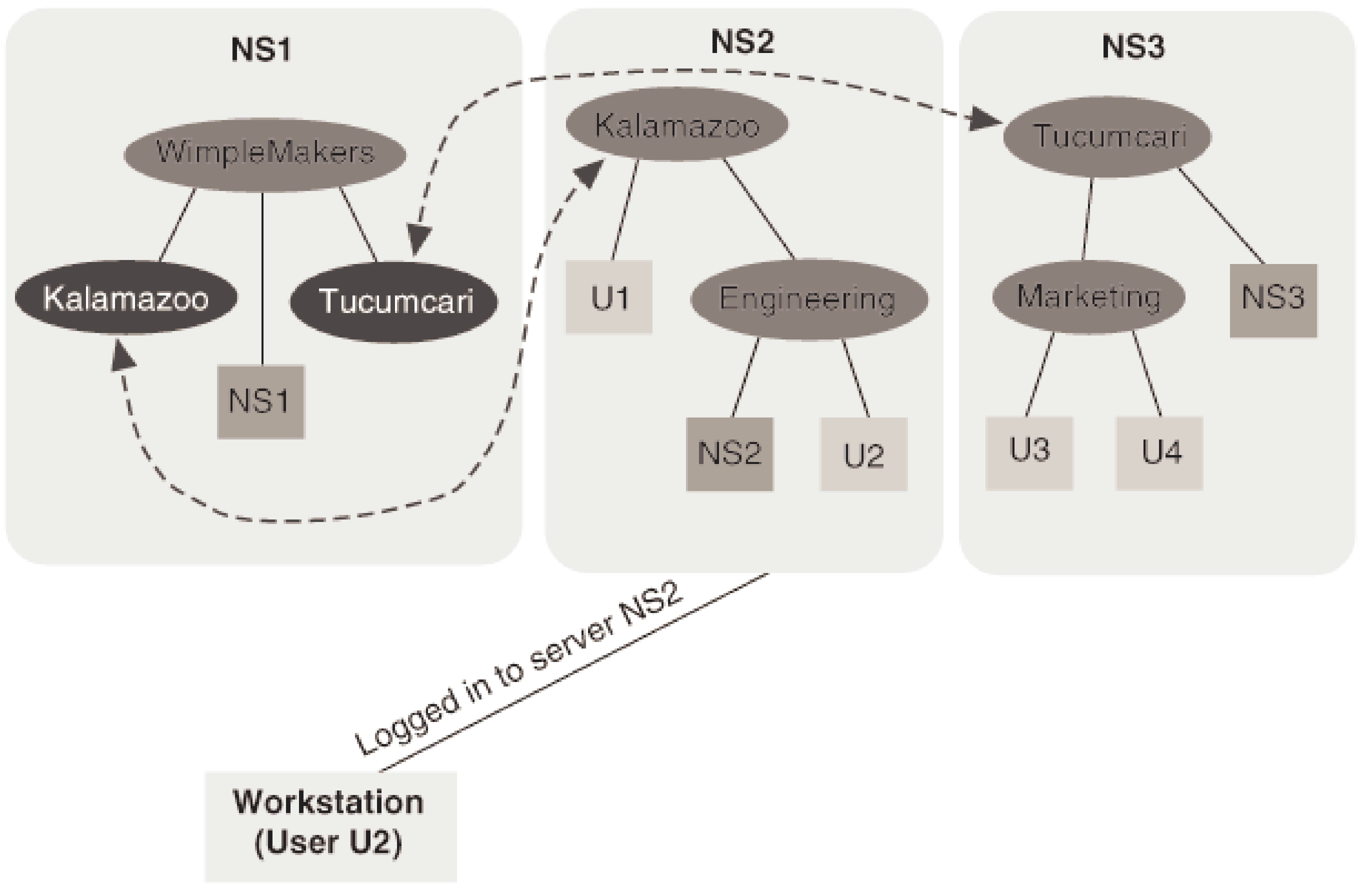
Figure 26. Sample Tree Structure



For this tree it was decided to make 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. (See the following figure.)

Figure 27. A Walk Up the Tree



This 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 NDS security ACLs.

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. Perhaps two or more servers have replicas of the Tucumcari partition.

Related Topics:

Progress Reports

NDS Partition: Guides

Trustees

The administrator can create ACL values to assign rights or privileges to an object or attributes. These are known as trustee assignments. NDS uses these trustee assignments in conjunction with the Inheritance ACLs and Inheritance Masks to compute the effective rights a given subject has on a given object.

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

Trustee assignments made to objects flow down the NDS 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". 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.

Trustee assignments made for attributes do not flow down the tree unless the assignment is for "all attributes". Assignments made to specifically designated individual attributes do not apply to subordinate objects or attributes of those subordinate objects.

Related Topics:

Equivalence in NDS

Access Control Lists

NDS Security: Concepts Guide

Typeless Names

Typeless Names are Context Names that lack the descriptors (O, OU, CN, etc). Context names are set to typeless with DCV_TYPELESS_NAMES (see Context Flags).

Setting the typeless context flag to ON allows your application to display nice looking names without types instead of the confusing look of names with types. However, if you set the context flags so that the libraries return typeless names, you can run into some problems with fully distinguished names.

The NDS libraries can add types to any components of a name that are passed into the API without a type. We learn from the Default Typing Rule that the most significant component (right-most) is an Organization (O). This can cause problems if a name like "JRoss.Engineering.US" is passed where US is a Country not an Organization. The libraries translate this name to "CN=JRoss.OU=Engineering.O=US", which is wrong. The call fails with a 0xFDA7 ERR_NO_SUCH_ENTRY.

However, if the context is set to "Engineering.US" and you pass in the partial name "JRoss", the call does not fail. The only time the libraries have problems with types is when they are passed fully Distinguished Names with no types. The only way the libraries can resolve the name is to use the Default Typing Rules.

To summarize, there are three areas where working with typeless names can cause problems. Here is a list of all three. Note how similar these look to the Default Typing Rule.

If you are working with leaf objects that don't use the attribute CN as the naming attribute.

If you are working with containers that don't use the naming attribute OU.

If your root object does not use the naming attribute O.

Related Topics:

Canonicalized Names

Attribute Type Abbreviations

NDS Context: Guides

Workstation Applications for NDS

The following table shows the major NDS operations that require application utility programs at the workstation. Each item represents classes of operations, and is not necessarily performed by a specific application or utility program.

Table auto. NDS Operation Types

Utility	**Function**
NDS Access	Access operations provide the principal NDS interface for users and administrators. Access operations create, retrieve, and maintain NDS objects and set access rights. These operations also allow users to search the NDS "yellow-page" style.
Partition Management	Partition management operations include adding and removing partitions of NDS. They also include creating and removing partition replicas and initiating synchronization processes.
Schema	Schema management operations allow

Management	administrators to extend the class and attribute definitions that define the information stored in NDS. New classes and attributes can be added to the base schema. In addition, optional attributes can be added to existing classes in the base schema. However, the class definitions included in the base schema (as shipped) cannot be deleted.
Backup	Backup operations maintain backup copies of NDS information base so that this information can be restored in the event of system failure.
Login	Login operations obtain the client's private key and use it to create the client signature and enter authenticated dialogues.
Logout	Logout operations remove the client's authentication data from the workstation and terminate the client's service connections.

Related Topics:

Application Programming Interfaces for NDS

NDS Workstation Components

NDS: Functions

NWDSAbbreviateName

Converts a NDS name (including the naming attributes) to its shortest form relative to a specified name context

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsname.h>
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSAbbreviateName (
   NWDSContextHandle   context,
   pnstr8              inName,
   pnstr8              abbreviatedName);
```

Pascal Syntax

```
#include <nwdsname.inc>

Function NWDSAbbreviateName
   (context : NWDSContextHandle;
    inName : pnstr8;
    abbreviatedName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

inName

(IN) Points to the object name to be abbreviated.

abbreviatedName

(OUT) Points to the abbreviated form of the name.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The caller must allocate space for the abbreviated name. The size of the allocated memory is ((MAX_RDN_CHARS)+1)*sizeof(character size), where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double byte). One character is used for NULL termination.

If the context flag associated with DCV_TYPELESS_NAMES is set on, the types are removed where possible. For example, the name

```
CN=Elmer Fudd.OU=Looney Tunes.O=Acme (context OU=Looney Tunes.O=Acme)
```

converts to

```
Elmer Fudd.
```

If the context flag associated with DCV_TYPELESS_NAMES is set off, the name converts to

```
CN=Elmer Fudd.
```

NCP Calls

None

See Also

NWDSCanonicalizeName

NWDSAbortPartitionOperation

Aborts a partition operation in progress
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1, 4.11
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSAbortPartitionOperation (
   NWDSContextHandle   context,
   pnstr8             partitionRoot);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSAbortPartitionOperation
  (context : NWDSContextHandle;
   partitionRoot : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

partitionRoot

(IN) Points to the root object name for the partition.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Two examples of partition operations are operations splitting a partition and joining a partition.

If possible, **NWDSAbortPartitionOperation** returns the partition to its state prior to the partition operation. If the partition operation cannot be aborted, **NWDSAbortPartitionOperation** returns ERR_CANNOT_ABORT.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSJoinPartitions, **NWDSSplitPartition**

NWDSAddFilterToken

Adds a node to the search filter expression tree

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

NWDSCCODE N_API NWDSAddFilterToken (
   pFilter_Cursor_T   cur,
   nuint16            tok,
   nptr               val,
   nuint32            syntax);
```

Pascal Syntax

```
#include <nwdsfilt.inc>

Function NWDSAddFilterToken
  (cur : pFilter_Cursor_T;
   tok : nuint16;
   val : nptr;
   syntax : nuint32
) : NWDSCCODE;
```

Parameters

cur

(IN) Points to a Filter_Cursor_T, which defines the current insertion point in the filter expression tree.

tok

(IN) Specifies the token to be added to the filter expression tree.

val

(IN) Points to either the attribute name or the attribute value, according to the token being added.

syntax

(IN) Specifies the attribute syntax associated with the *val* parameter.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Each node contains a token and a syntax with an associated value. The token is identified by *tok* with the following values:

```
FTOK_END         0
FTOK_OR          1
FTOK_AND         2
FTOK_NOT         3
FTOK_LPAREN      4
FTOK_RPAREN      5
FTOK_AVAL        6
FTOK_EQ          7
FTOK_GE          8
FTOK_LE          9
FTOK_APPROX     10
FTOK_ANAME      14
FTOK_PRESENT    15
FTOK_RDN        16
FTOK_BASECLS    17
FTOK_MODTIME    18
FTOK_VALTIME    19
```

The relationship between the *tok*, *val*, and *syntax* parameters is as follows:

If the *tok* parameter is FTOK_ANAME (meaning attribute name), the *val* parameter must point to a copy of the attribute name, and the *syntax* parameter must be set to the appropriate attribute syntax ID.

If the *tok* parameter is FTOK_AVAL (meaning attribute value), the *val* parameter must point to a copy of the attribute value, and the *syntax* parameter must be set to the appropriate attribute syntax ID.

If the *tok* parameter is neither FTOK_ANAME or FTOK_AVAL, the *val* and *syntax* parameters are ignored and can be set to NULL.

The *val* parameter must point to a dynamically allocated memory buffer that can be freed by calling either the **NWDSPutFilter** or **NWDSAddFilterToken** function.

The **NWDSPutFilter** function frees up the memory associated with the expression tree. However, if **NWDSAddFilterToken** returns an error while you are creating an expression tree, you should not call the **NWDSPutFilter** function, but call the **NWDSFreeFilter** function to free up the memory associated with the expression tree.

The *expect* field of the *cur* parameter contains a bit-map representation of the valid token at the current position in the tree. The *tok* parameter must correspond to one of these tokens. If **NWDSAddFilterToken** returns SUCCESSFUL, the *expect* field is updated according to the next position in the tree (the insertion point of the next token).

Parsing of the token expression list is performed by **NWDSAddFilterToken**.

For information about how to conduct a search and for more details about NDS searches, see Searching NDS and NDS Search Introduction.

NCP Calls

None

See Also

NWDSAllocFilter, **NWDSDelFilterToken**, **NWDSFreeFilter**, **NWDSPutFilter**

NWDSAddObject

Adds an object to the NDS tree
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSAddObject (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnint32             iterationHandle,
   nbool8              more,
   pBuf_T              objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSAddObject
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint32;
   more : nbool8;
   objectInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object to be added.

iterationHandle

(IN) Points to the iteration number (-1 initially).

more

(IN) Specifies whether additional information will be returned:

0 No more information

0 No more information

nonzero More information will be returned

objectInfo

(IN) Points to a request buffer containing the attribute values for the new object.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

For **NWDSAddObject** to succeed, the new object's immediate superior must already exist.

objectName identifies the name of the object to be added. For example,

```
CN=Elmer Fudd.OU=Looney Tunes.O=Acme.C=US.
```

The object can be an alias entry.

NOTE: If the *iterationHandle* parameter is set to 0 initially, **NWDSAddObject** will ignore the value and process the request as if -1 was passed.

If the *more* parameter is set to nonzero, **NWDSAddObject** will perform the necessary steps to iteratively call itself.

In order to iteratively call **NWDSAddObject**, the DS.NLM file must support the iteration feature or ERR_BUFFER_FULL will be returned.

All of an object's mandatory attributes must be supplied for **NWDSAddObject** to succeed. For example, Object Class is a mandatory attribute for any object that is added. This is the base class of the object.

While NDS ensures that new objects conform to the NDS Schema, if an alias is being created for an existing object, no check is made to ensure the alias's Aliased Object Name attribute points to a valid object. This check occurs on the server when the aliased object name is translated to a local ID.

NWDSAddObject never dereferences aliases. The setting of the context flag associated with DCV_DEREF_ALIASES in the context field associated with DCK_FLAGS is ignored.

For more information, see Adding an NDS Object.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSListContainableClasses, **NWDSModifyObject**, **NWDSRemoveObject**

NWDSAddPartition (obsolete 9/97)

Creates the root object of a new NDS partition but is now obsolete. Call **NWDSAddReplica** and **NWDSSplitPartition** instead.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSAddPartition (
   NWDSContextHandle   context,
   pnstr8              server,
   pnstr8              partitionRoot,
   pnint32             iterationHandle,
   nbool8              more,
   pBuf_T              objectInfo);
```

Pascal Syntax

```
Function NWDSAddPartition
  (context : NWDSContextHandle;
   server : pnstr8;
   partitionRoot : pnstr8;
   iterationHandle : pnint32;
   more : nbool8;
   objectInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

server

(IN) Points to the name of the server where the partition is to be added.

partitionRoot

(IN) Points to the name of the root object of the new partition. (New object being created.)

iterationHandle

(IN/OUT) Is reserved; pass in 0.

more

(IN) Is reserved; pass in NULL.

objectInfo

(IN) Points to the attributes that define the root object of the new partition.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The initial partition replica will always be of type RT_MASTER, which is enumerated in NWDSDEFS.H.

The *server* parameter identifies the server where the master replica of the new partition is to be stored.

The *objectInfo* parameter points to the Buf_T structure containing attribute information, which, together with information from the partition root name, constitutes the object to be created. For more information, see Adding a Partition.

The type of attribute information needed to create the partition object is listed in NDS Object Class Definitions. The user can add any attributes to the partition object that are not read only. Values for other partition attributes are assigned automatically.

Aliases are never dereferenced by **NWDSAddReplica**. The setting of the NDS context flag associated with DCV_DEREF_ALIASES is not relevant and is ignored.

You cannot call **NWDSAddObject** to create a partition; you must call either **NWDSAddPartition (obsolete 9/97)** or **NWDSSplitPartition** to do so.

Additional replicas of a partition can be added to the system by calling **NWDSAddReplica**.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAddReplica, **NWDSJoinPartitions**, **NWDSListPartitions**, **NWDSRemovePartition**, **NWDSSplitPartition**

NWDSAddReplica

Adds a replica of an existing NDS partition to a server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSAddReplica (
   NWDSContextHandle   context,
   pnstr8              server,
   pnstr8              partitionRoot;
   nuint32             replicaType);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSAddReplica
  (context : NWDSContextHandle;
   server : pnstr8;
   partitionRoot : pnstr8;
   replicaType : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

server

(IN) Points to the name of the server where the replica is to be stored.

partitionRoot

(IN) Points to the name of the root object of the NDS partition to be replicated.

replicaType

(IN) Specifies the type of the new replica (secondary or read-only).

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The partition must be created beforehand by calling **NWDSAddReplica** or **NWDSSplitPartition**.

replicaType determines the type of replica to be created:

1	RT_SECONDARY	Secondary
2	RT_READONLY	Read-only

NOTE: You cannot create a master replica type (RT_MASTER) with **NWDSAddReplica**.

Aliases are never dereferenced by **NWDSAddReplica**. The setting of the NDS context flag associated with DCV_DEREF_ALIASES is not relevant and is ignored.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info

0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAddReplica, **NWDSChangeReplicaType**, **NWDSRemoveReplica**, **NWDSSplitPartition**

NWDSAddSecurityEquiv

Adds to the specified object's security equivalence
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE NWDSAddSecurityEquiv (
   NWDSContextHandle   context,
   pnstr8             *equalFrom,
   pnstr8             *equalTo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSAddSecurityEquiv
  (context : NWDSContextHandle;
   equalFrom : pnstr8;
   equalTo : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

equalFrom

(IN) Points to the name of the object that will receive security equivalence.

equalTo

(IN) Points to the name to be added to the Security Equivalence attribute of the object specified by *equalFrom*.

Return Values

0x0000	SUCCESSFUL

negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

If **NWDSAddSecurityEquiv** is successful, it will place the name of the object specified by *equalTo* into the Security Equals attribute of the object specified by the *equalFrom* parameter. (Security Equals is a multivalued attribute.)

If the object specified by the *equalFrom* parameter does not contain sufficient rights to add the security equivalence to its list, **NWDSAddSecurityEquiv** will return ERR_NO_ACCESS.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRemSecurityEquiv

NWDSAllocBuf

Allocates a Buf_T for use as a request or result buffer parameter to a NDS function

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSAllocBuf (
   size _t     size,
   ppBuf_T   buf);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSAllocBuf
  (size : size_t;
   buf : ppBuf_T
) : NWDSCCODE;
```

Parameters

size

(IN) Specifies the number of bytes to allocate to the buffer.

buf

(OUT) Points to Buf_T containing the memory allocated for the buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The following two message sizes are defined in NWDSDC.H:

4096	DEFAULT_MESSAGE_LEN
64512	MAX_MESSAGE_LEN

The total bytes allocated for the buffer is *size*+sizeof(Buf_T).

For most operations, the size of DEFAULT_MESSAGE_LEN can be used. It is up to the developer to determine by experimentation if another size optimizes an application's performance.

When determining a buffer size, keep in mind the effects of buffer size. A smaller buffer means multiple iterations of an operation might need to be performed to retrieve all of the operation's results. On the other hand, using a large buffer might allow the operation to be completed in one step, but cause a significant delay for the user.

If **NWDSAllocBuf** is successful, *buf* is set to point to the allocated buffer.

NCP Calls

None

See Also

NWDSFreeBuf, **NWDSInitBuf**

NWDSAllocFilter

Allocates a filter expression tree and initializes a cursor to the current insertion point

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

NWDSCCODE N_API NWDSAllocFilter (
   ppFilter_Cursor_T   cur);
```

Pascal Syntax

```
#include <nwdsfilt.inc>

Function NWDSAllocFilter
   (cur : ppFilter_Cursor_T
) : NWDSCCODE;
```

Parameters

cur

(IN/OUT) Points to the current filter-cursor position in the allocated filter.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

NCP Calls

None

See Also

NWDSAddFilterToken

NWDSPutFilter

NWDSAuditGetObjectID

Returns a connection handle and an object ID for the object name

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsaud.h>

NWDSCCODE N_API NWDSAuditGetObjectID (
   NWDSContextHandle      context,
   pnstr8                 objectName,
   NWCONN_HANDLE N_FAR   *conn,
   pnuint32               objectID);
```

Pascal Syntax

```
#include <nwdsaud.inc>

Function NWDSAuditGetObjectID
  (context : NWDSContextHandle;
   objectName : pnstr8;
   Var conn : NWCONN_HANDLE;
   objectID : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the directory context for the request.

objectName

(IN) Points to the name of the object to get the ID for.

conn

(OUT) Points to the connection handle where the object resides.

objectID

(OUT) Points to the NDS object ID.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The returned connection handle is the NetWare server where the object is stored.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAddObject, **NWDSResolveName**

NWDSAuthenticate

Establishes an authenticated connection to a secured NetWare® server using the unauthenticated connection and local data cached by calling **NWDSLogin**

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsasa.h>

NWDSCCODE N_API NWDSAuthenticate (
   NWCONN_HANDLE         conn,
   nflag32               optionsFlag,
   pNWDS_Session_Key_T   sessionKey);
```

Pascal Syntax

```
#include <nwdsasa.inc>

Function NWDSAuthenticate
  (conn : NWCONN_HANDLE;
   optionsFlag : nflag32;
   sessionKey : pNWDS_Session_Key_T
) : NWDSCCODE;
```

Parameters

conn

(IN) Specifies the client's initial unauthenticated connection handle.

optionsFlag

(IN) Reserved; pass in a zero (0).

sessionKey

(IN) Reserved; pass in NULL.

Return Values

0x0000	SUCCESSFUL

negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSAuthenticate first checks to see if the specified connection is authenticated. If the connection is authenticated, **NWDSAuthenticate** will return SUCCESFUL and end the call.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAuthenticateConn, **NWDSLogin**

NWDSAuthenticateConn

Establishes an authenticated connection to a secured NetWare server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSAuthenticateConn (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle);
```

Pascal Syntax

```
#include <nwdsasa.inc>

Function NWDSAuthenticateConn
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(IN) Points to the connection handle to authenticate.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSAuthenticateConn establishes an authenticated connection to a secured NetWare server using the unauthenticated connection and the secured NetWare server using the unauthenticated connection and the identity, established by calling **NWDSLogin**, of the object derived from the context handle.

The context handle is used to indicate which tree (and hence which identity) to use in authenticating the connection handle. If the underlying requester does not support multiple tree authentications, the tree value of the context handle is ignored.

NWDSAuthenticateConn first checks to see if the specified connection is authenticated. If the connection is authenticated, **NWDSAuthenticateConn** will return SUCCESFUL and end the call.

NCP Calls

0x2222 104 2   Send NDS Fragmented Request/Reply

See Also

NWDSOpenConnToNDSServer, **NWDSOpenMonitoredConn, NWCCOpenConnByName**

NWDSBackupObject

Backs up the attribute names and values for an object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE NWDSBackupObject (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnint32             iterationHandle,
   pBuf_T              objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSBackupObject
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint32;
   objectInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object for which information is to be returned.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSBackupObject**.

objectInfo

(OUT) Points to the requested attribute names and values.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSBackupObject is used to back up the attributes and attribute values for one object at a time. To back up NDS, call the **NWDSBackupObject** function for each object.

iterationHandle is used to control retrieval of results that are larger than the result buffer supplied by *objectInfo*.

Before the initial call to **NWDSBackupObject**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

When **NWDSBackupObject** returns from its initial call, if the result buffer holds the complete results, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS on return. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSBackupObject** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

CAUTION: The information returned in *objectInfo* must be stored so it can be passed to NWDSRestoreObject in the expected manner. NWDSRestoreObject expects an nuint32 array pointer and an nuint8 pointer specifying the length of the information to be restored.

Each time **NWDSBackupObject** is called, save the number of bytes returned by *objectInfo*->*curLen* starting from the address pointed to by *objectInfo*->*data*. *objectInfo* must be worked with directly; there are no NDS functions that will retrieve this information.

It is the developer's responsibility to decide how to store the information so it can be restored when calling **NWDSRestoreObject**.

The results of **NWDSBackupObject** are not ordered. Attribute information might not be stored in the result buffer in alphabetical order.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRestoreObject

NWDSBeginClassItem

Begins a class item definition (which is a part of an object class definition) in a request buffer to be used by a NDS Schema function

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSBeginClassItem (
   NWDSContextHandle   context,
   pBuf_T              buf);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSBeginClassItem
  (context : NWDSContextHandle;
   buf : pBuf_T
) : NWDSCCODE
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the request buffer being prepared.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

buf points to a Buf_T, which is allocated by **NWDSAllocBuf** and initialized by **NWDSInitBuf** for the DSV_DEFINE_CLASS operation.

NWDSBeginClassItem is used in conjunction with **NWDSPutClassName** and **NWDSPutAttrName** to prepare a request buffer for **NWDSDefineClass** to use in creating a new object-class definition. This request buffer must contain a sequence of five sets of class definition item lists. The lists must occur in the following order:

1. Super Class Names
2. Containment Class Names
3. Naming Attribute Names
4. Mandatory Attribute Names
5. Optional Attribute Names

If a particular definition item list is empty, **NWDSBeginClassItem** must still be called for that list. For example, if the class definition has no mandatory attributes, you must call **NWDSBeginClassItem** to move to the Mandatory Attribute Names list and then immediately call **NWDSBeginClassItem** again to move to the Optional Attribute Names list.

The complete steps for creating a new object class definition are found in the reference for **NWDSDefineClass**.

NCP Calls

None

See Also

NWDSPutClassName, **NWDSPutClassItem**

NWDSCanDSAuthenticate

Determines if NDS credentials exist for the specified tree name
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSCanDSAuthenticate (
   NWDSContextHandle   context);
```

Pascal Syntax

```
#include <nwdsconn.inc>

Function NWDSCanDSAuthenticate
  (context : NWDSContextHandle
) : NWDSCCODE;
```

Parameters

context
(IN) Specifies the NDS context for the request.

Return Values

These are common return values; see Return Values for more information.

0	No credentials found
1	Credentials exist

Remarks

NWDSCanDSAuthenticate is similar to **NWIsDSAuthenticated**, but is enabled for multiple tree environments. **NWDSCanDSAuthenticate** indicates if NDS credentials exist for the tree name described in the context. If credentials exist for the tree, authentication can be performed to servers within the tree.

NCP Calls

None

See Also

NWDSAuthenticateConn, **NWIsDSAuthenticated**

NWDSCanonicalizeName

Converts an abbreviated name to the canonical form

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsname.h>

NWDSCCODE N_API NWDSCanonicalizeName (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              canonName);
```

Pascal Syntax

```
#include <nwdsname.inc>

Function NWDSCanonicalizeName
  (context : NWDSContextHandle;
   objectName : pnstr8;
   canonName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the object name to be expressed in canonical form.

canonName

(OUT) Points to the canonical form of the name.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The canonical form of a name includes the full path of the name (a complete name) with the naming attribute type specification for each naming component. Standard naming attribute type abbreviations are used where available. In addition, multiple white spaces are removed from the name.

For example, if the input is

```
CN=Elmer Fudd
```

and the name context is

```
OU=Looney Toons.O=Acme
```

the canonicalized name is

```
CN=Elmer Fudd.OU=Looney Toons.O=Acme
```

The canonicalized name will always contain types, regardless of the setting of the context flag associated with DCV_TYPELESS_NAMES.

objectName supplies the abbreviated form of a NDS name. The name can be typed or typeless. It can also be truncated. It is assumed that a truncated name is relative to the naming path supplied by the specified context.

canonName receives the canonical form of the name. The caller must allocate space for the canonicalized name. The size of the allocated memory is ((MAX_DN_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double byte). One character is used for NULL termination.

NCP Calls

None

See Also

NWDSAbbreviateName

NWDSChangeObjectPassword

Changes the authentication password for an NDS object once a public/private key pair has been assigned

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsasa.h>

NWDSCCODE N_API NWDSChangeObjectPassword (
   NWDSContextHandle   context,
   nflag32             optionsFlag,
   pnstr8              objectName,
   pnstr8              oldPassword,
   pnstr8              newPassword);
```

Pascal Syntax

```
#include <nwdsasa.inc>

Function NWDSChangeObjectPassword
  (context : NWDSContextHandle;
   optionsFlag : nflag32;
   objectName : pnstr8;
   oldPassword : pnstr8;
   newPassword : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

optionsFlag

(IN) Reserved; pass in zero.

objectName

(IN) Points to the object name whose password is to be changed.

oldPassword

(IN) Points to the object's current password.

newPassword

(IN) Points to the object's new password.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

For **NWDSChangeObjectPassword** to succeed, *oldPassword* must be correct. If no value is currently assigned to the password, *oldPassword* should point to a zero-length string.

If no new password value is desired, *newPassword* should point to a zero-length string.

A password can be any length and all characters are significant. Uppercase and lowercase characters are distinct.

If an application has a local copy of any password value, the value should be erased as soon as possible to prevent compromising the security of the password.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSGenerateObjectKeyPair

NWDSChangeReplicaType

Changes the replica type of a given replica on a given server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSChangeReplicaType (
   NWDSContextHandle   context,
   pnstr8              replicaName,
   pnstr8              server,
   nuint32             newReplicaType);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSChangeReplicaType
  (context : NWDSContextHandle;
   replicaName : pnstr8;
   server : pnstr8;
   newReplicaType : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

replicaName

(IN) Points to the root object name of the NDS partition whose replica type will be changed.

server

(IN) Points to the name of the server on which the replica resides.

replicaType

(IN) Specifies the replica type the given replica is to be changed to.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

replicaType can be one of the following types enumerated in NWDSDEFS.H:

0	RT_MASTER	Master replica
1	RT_SECONDARY	Secondary replica
2	RT_READONLY	Read-only replica

A change in type from read-only to secondary or secondary to read-only affects only the given replica. A change to RT_MASTER results in the current master being changed to a secondary replica.

The replica type of the master may not be changed directly by calling **NWDSChangeReplicaType**. The replica type of the master replica can only change as a side effect of **NWDSChangeReplicaType** changing another replica's type to RT_MASTER.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)

0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSSplitPartition, **NWDSJoinPartitions**, **NWDSAddReplica**, **NWDSRemoveReplica**

NWDSCIStringsMatch

Tests two case ignore strings (defined by *CI_String_T*) to determine if the two strings are equivalent

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsname.h>

NWDSCCODE N_API NWDSCIStringsMatch (
   NWDSContextHandle   context,
   pnstr8              string1,
   pnstr8              string2,
   pnint               matches;)
```

Pascal Syntax

```
#include <nwdsname.inc>

Function NWDSCIStringsMatch
  (context : NWDSContextHandle;
   string1 : pnstr8;
   string2 : pnstr8;
   matches : pnint
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context to be used. It is created by calling **NWDSCreateContextHandle**.

string1

(IN) Points to the first string to compare.

string2

(IN) Points to the second string to compare.

matches

(OUT) Points to a boolean indicating whether the strings match: 0 = Don't match; 1 = Match.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0xFE0D	UNI_NO_DEAFAULT
0xFE0F	UNI_HANDLE_MISMATCH
0xFE10	UNI_HANDLE_BAD
0xFED1	ERR_BAD_CONTEXT
0xFED3	ERR_NOT_ENOUGH_MEMORY

Remarks

Case Ignore String is a syntax used by some of the attributes (such as CN, Description, NDS, Services flags, and Title) for NDS objects.

Depending on the context, **NWDSCIStringsMatch** compares two Unicode strings in the local or Unicode code page. This function ignores leading and trailing white space, which is either " " (space, 0x0020) or "_" (underscore, 0x005F). Also, it matches any consecutive internal white space, regardless of quantities. For example, if the string has a single internal white space character and another has five, **NWDSCIStringsMatch** matches the strings. Finally, **NWDSCIStringsMatch** ignores case in comparisons.

NWDSCIStringsMatch is a local function.

NCP Calls

None

NWDSCloseIteration

Frees memory associated with an iteration handle in the event the client chooses to discontinue iterative calls to the server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsmisc.h>
#include <nwdsdefs.h>

NWDSCCODE N_API NWDSCloseIteration (
   NWDSContextHandle   context,
   int32              iterationHandle,
   uint32             operation);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSCloseIteration
  (context : NWDSContextHandle;
   iterationHandle : nint32;
   operation : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

iterationHandle

(IN) Specifies the iteration handle previously received from the server.

operation

(IN) Specifies the NDS operation associated with *iterationHandle*.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSCloseIteration is called to discontinue an iterative operation, such as Read, List, and Search, before the operation is complete. In the event the client chooses to discontinue the iterative exchange with the server, **NWDSCloseIteration** frees memory on both the client and the server and states information associated with the handle.

Functions such as **NWDSList**, **NWDSRead**, and **NWDSSearch** free the memory and state information associated with an operation when they return with *iterationHandle* set to NO_MORE_ITERATIONS. **NWDSCloseIteration** is called to stop the operation before these functions set *iterationHandle* to NO_MORE_ITERATIONS.

operation tags follow:

3	DSV_READ	**NWDSExtSyncRead** **NWDSListAttrsEffectiveRights** **NWDSRead** **NWDSReadReferences**
4	DSV_COMPARE	**NWDSCompare**
6	DSV_SEARCH	**NWDSExtSyncList** **NWDSExtSyncSearch** **NWDSListByClassAndName** **NWDSListContainers** **NWDSPutFilter** **NWDSSearch**
7	DSV_ADD_ENTRY	**NWDSAddObject**

9	DSV_MODIFY_ENTRY	**NWDSModifyObject**
12	DSV_READ_ATTR_DEF	**NWDSReadAttrDef**
14	DSV_DEFINE_CLASS	**NWDSDefineClass**
15	DSV_READ_CLASS_DEF	**NWDSReadClassDef**
16	DSV_MODIFY_CLASS_DEF	**NWDSModifyClassDef**
18	DSV_LIST_CONTAINABLE_CLASSES	**NWDSListContainableClasses**
22	DSV_LIST_PARTITIONS	**NWDSListPartitions**
45	DSV_BACKUP_ENTRY	**NWDSBackupObject**
46	DSV_RESTORE_ENTRY	**NWDSRestoreObject**

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRead, **NWDSList**, **NWDSSearch**, **NWDSListAttrsEffectiveRights**, **NWDSBackupObject**, **NWDSRestoreObject**, **NWDSListPartitions**, **NWDSListContainableClasses**, **NWDSReadAttrDef**, **NWDSReadClassDef**

NWDSCompare

Compares an object's attribute value with a specified value
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSCompare (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pBuf_T              buf,
   pnbool8             matched);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSCompare
  (context : NWDSContextHandle;
   objectName : pnstr8;
   buf : pBuf_T;
   matched : pnbool8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object whose attribute is being compared.

buf

(IN) Points to a request buffer containing the attribute name and value to be compared with the object's attribute value.

matched

(OUT) Points to a boolean value indicating the result of the comparison.

Return Values

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The comparison is in the form of "attribute name=attribute value." For example, the attribute name "Description" and the value "PostScript" might be used to determine if a particular printer's page description language is PostScript.

matched receives a Boolean indicating the result of the comparison. The result is TRUE if the comparison was successful, otherwise the result is FALSE.

For more information, see Comparing Attribute Values.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRead

NWDSComputeAttrValSize

Computes, in conjunction with **NWDSGetAttrVal**, the size of the attribute value at the current position in the result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSComputeAttrValSize (
   NWDSContextHandle   context,
   pBuf_T              buf,
   nuint32             syntaxID,
   pnuint32            attrValSize);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSComputeAttrValSize
  (context : NWDSContextHandle;
   buf : pBuf_T;
   syntaxID : nuint32;
   attrValSize : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to a result buffer positioned at an attribute value.

syntaxID

(IN) Specifies the numeric ID of the attribute value.

attrValSize

(OUT) Points to the size (in bytes) required to retrieve the attribute.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Since Buf_T buffers are opaque to client applications, a client cannot view a result buffer directly to see the size of the values returned in the buffer. Call **NWDSComputeAttrValSize** to find the size and syntax of the current attribute value in the buffer and then dynamically allocate memory of that size to hold the current attribute's value. Then retrieve the value by calling **NWDSGetAttrVal**.

Call **NWDSComputeAttrValSize** once for each attribute value you retrieve from the result buffer.

syntaxID identifies the syntax data type the attribute information is stored in. The data structures associated with the syntaxes are listed inNDS Attribute Syntax Definitions. The enumerated types for syntaxes (such as SYN_DIST_NAME) are located in NWDSDEFS.H.

attrValSize points to the size of the attribute value in bytes. This size can be used as input to a memory allocation request. The size is large enough to contain the attribute value along with any structure returned by **NWDSGetAttrVal**.

NCP Calls

None

See Also

NWDSGetAttrVal

NWDSCreateContext (obsolete 03/97)

Creates a NDS context for NDS client operations and initializes it to the default configuration but is now obsolete. Call NWDSCreateContextHandle instead.

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdc.h>

NWDSContextHandle N_API NWDSCreateContext (
   void);
```

Pascal Syntax

```
#include <nwdsdc.inc>

Function NWDSCreateContext
  : NWDSContextHandle;
```

Return Values

These are common return values; see Return Values for more information.

0x0000 or higher	A handle to a newly-created context
0xFEB8	ERR_CONTEXT_CREATION

Remarks

The maximum number of contexts that an application can create depends upon the application's platform. The maximum number of contexts is 16 for DOS, Windows95 and Windows NT; the limit is 48 for Windows and OS/2. The maximum contexts for NLM applications is limited only by the available memory on the server.

If successful, **NWDSCreateContext (obsolete 03/97)** returns a handle for the newly-created context. The range of values for this handle is from 0 to the platfom's limit, minus one. For example, the range for DOS is 0 to 7.

NOTE: **NWDSCreateContext (obsolete 03/97)** can fail for the following reasons: (1) insufficient memory to create the context, (2) maximum number of contexts exhausted (such as 8 for DOS), and (3) Unicode tables were not initialized (see **NWInitUnicodeTables**).

The following list shows the default values set when a NDS context is created:

The flags in the context variable associated with DCK_FLAGS are set as follows:

```
The flag associated with DCV_ASYNC_MODE is set off. (It is res
The flag associates with DCV_CANONICALIZE_NAME is set on.
The flag associated with DCV_DEREF_ALIASES is set on.
The flag associated with DCV_TYPELESS_NAMES is set off.
The flag associated with DCV_XLATE_STRINGS is set on.
The flag associated with DCV_DEREF_BASE_CLASS is set off.
The flag associated with DCV_DISALLOW_REFERRALS is set off.
```

The context variable associated with DCK_CONFIDENCE is set to DCV_LOW_CONF.

The context variable associated with DCK_NAME_CONTEXT is set to the current global name context which is obtained from NET.CFG for workstations, and from the Bindery emulation context setting for NLM applications. (Changing the setting of the name context does not change the value in the NET.CFG file for workstations or the value of the Bindery emulation context on servers.)

The context variable associated with DCK_TRANSPORT_TYPE is set to NT_IPX. For now this field is reserved.

The context variable associated with DCK_REFERRAL_SCOPE is set to DCV_ANY_SCOPE.

The context variable associated with DCK_LAST_CONNECTION is initially cleared. It is then set to the connection handle of the last server to which the library sent a request.

The context variable associated with DCK_TREE_NAME is set to the name of the tree in the current context.

The structure of the NDS context is not available for the developer to manipulate directly. Instead, NDS context variables are modified by calling **NWDSSetContext**. The current values of NDS context variables are retrieved by calling **NWDSGetContext**.

NCP Calls

None

See Also

NWDSCreateContextHandle, NWDSDuplicateContext, NWDSFreeContext, NWDSGetContext, NWDSSetContext

NWDSCreateContextHandle

Allocates memory for a new context structure and initializes it with default values

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsdc.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSCreateContextHandle (
   NWDSContextHandle N_FAR  *newHandle);
```

Pascal Syntax

```
#include <nwdsdc.inc>

Function NWDSCreateContextHandle
  (Var newHandle : NWDSContextHandle
) : NWDSCCODE;
```

Parameters

newHandle

(OUT) Points to the newly created context handle.

Return Values

0x0000	SUCCESSFUL
nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSCreateContextHandle allocates storage for a new context structure, initializes the structure with default values, and returns a handle to reference the context structure.

NCP Calls

None

See Also

NWDSDuplicateContextHandle, **NWDSFreeContext**

NWDSDefineAttr

Adds a new attribute definition to the NDS Schema
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdssch.h>

NWDSCCODE N_API NWDSDefineAttr (
   NWDSContextHandle   context,
   pnstr8              attrName,
   pAttr_Info_T        attrDef);
```

Pascal Syntax

```
#include <nwdssch.inc>

Function NWDSDefineAttr
  (context : NWDSContextHandle;
   attrName : pnstr8;
   attrDef : pAttr_Info_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

attrName

(IN) Points to the name for the new attribute.

attrDef

(IN) Points to the remaining information for the new attribute definition.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The name of the new attribute must be unique within the NDS Schema attribute definitions. The names of the attributes for the Base Schema are listed in NDS Attribute Type Definitions. New attributes added by other applications must be read from the Schema on a server by calling **NWDSReadAttrDef**.

New Attribute names should be cleared through Developer Support to guarantee uniqueness.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSDefineClass

NWDSDefineClass

Adds a new object class definition to the NDS Schema
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdssch.h>

NWDSCCODE N_API NWDSDefineClass (
   NWDSContextHandle   context,
   pnstr8              className,
   pClass_Info_T       classInfo,
   pBuf_T              classItems);
```

Pascal Syntax

```
#include <nwdssch.inc>

Function NWDSDefineClass
  (context : NWDSContextHandle;
   className : pnstr8;
   classInfo : pClass_Info_T;
   classItems : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

className

(IN) Points to the name of the new object class.

classInfo

(IN) Points to the class flags and ASN.1 ID for the new class.

classItems

(IN) Points to the remaining class definition.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The name of the new object class must be unique within the NDS Schema class definitions. The names of the classes for the Base Schema are listed in NDS Object Class Definitions. New object classes added by other applications must be read from the Schema on a server by calling **NWDSReadClassDef**.

New object-class names should be cleared through Developer Support to guarantee uniqueness.

classItems points to a request buffer containing additional information that defines the object. This buffer contains a sequence of five lists containing either class names or attribute names. The lists must occur in the following order.

1. Super Class Names
2. Containment Class Names
3. Naming Attribute Names
4. Mandatory Attribute Names
5. Optional Attribute Names

NCP Calls

0x2222 23 17 Get File Server Information

0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSDefineAttr, **NWDSModifyClassDef**

NWDSDelFilterToken

Deletes the most recently added token from a filter expression tree
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

NWDSCCODE N_API NWDSDelFilterToken (
   pFilter_Cursor_T       cur,
   void (N_FAR N_CDECL  *freeVal)(nuint32 syntax, nptr val);
```

Pascal Syntax

```
#include <nwdsfilt.inc>

Function NWDSDelFilterToken
  (cur : pFilter_Cursor_T;
   freeVal : FreeValProc
) : NWDSCCODE;
```

Parameters

cur

(IN) Points to the current insertion point in the filter expression tree.

freeVal

(IN) Points to the function to be used to free attribute values.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

freeVal is a pointer to a function freeing the attribute values. The function is passed the syntax ID and the address of the area to free. *freeVal* may be

NULL, in which case no attribute values are freed.

If **NWDSDelFilterToken** is successful, *cur* is updated to reflect the current position in the expression tree (the insertion point of the next token).

Syntax IDs (such as SYN_BOOLEAN) are enumerated in NWDSDEFS.H.

NCP Calls

None

See Also

NWDSAddFilterToken, **NWDSAllocFilter**, **NWDSFreeFilter**, **NWDSPutFilter**

NWDSDuplicateContext

Creates an NDS context and initializes it to the same settings as an existing NDS Context

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdc.h>

NWDSContextHandle N_API NWDSDuplicateContext (
   NWDSContextHandle   oldContext);
```

Pascal Syntax

```
#include <nwdsdc.inc>

Function NWDSDuplicateContext
  (oldContext : NWDSContextHandle
) : NWDSContextHandle;
```

Parameters

oldContext

(IN) Specifies the NDS context to duplicate.

Return Values

These are common return values; see Return Values for more information.

0x000 0	SUCCESSFUL
0xFEB 8	ERR_CONTEXT_CREATION

Remarks

If successful, **NWDSDuplicateContext** returns a value identifying the created NDS context. The newly created context will have a copy of the contents of the NDS context specified by *oldContext*. If *oldContext* does not reference a valid NDS context, the new context will be initialized with default values as in **NWDSCreateContextHandle**.

The advantage in calling **NWDSDuplicateContext** is that it copies the context settings of the existing context. If you are using context settings that are not the default, **NWDSDuplicateContext** lets you avoid making some additional calls to modify the default context settings.

NCP Calls

None

See Also

NWDSCreateContextHandle, **NWDSFreeContext**, **NWDSSetContext**, **NWDSGetContext**

NWDSDuplicateContextHandle

Allocates memory for a new context structure and initializes it with values copied from the source context structure

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsdc.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSDuplicateContextHandle (
   NWDSContextHandle           srcContextHandle,
   NWDSContextHandle  N_FAR   *destContextHandle);
```

Pascal Syntax

```
#include <nwdsdc.inc>

Function NWDSDuplicateContextHandle
  (Var destContextHandle : NWDSContextHandle;
   srcContextHandle : NWDSContextHandle
) : NWDSCCODE;
```

Parameters

srcContextHandle

(IN) Specifies the context handle referencing the structure to be duplicated.

destContextHandle

(OUT) Points to the newly created context handle.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSDuplicateContextHandle allocates storage for a new context structure and copies the values of the source context structure referenced by *srcContextHandle* to the newly allocated context structure. If the *srcContextHandle* is invalid, allocation of a new context structure is still attempted. In this case, the default values of **NWDSCreateContextHandle** will be used to initialize the new context structure.

NWDSDuplicateContextHandle differs from **NWDSDuplicateContext** in that the return code may now specify codes indicative of the reason for the failure.

NCP Calls

None

See Also

NWDSCreateContextHandle, **NWDSDuplicateContext**

NWDSExtSyncList

Lists the immediate subordinates for an NDS objects and places restrictions on the subordinates names, classes, modification times, and object types

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1, 4.11

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSExtSyncList (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              className,
   pnstr8              subordinateName,
   pnint32             iterationHandle,
   pTimeStamp_T        timeStamp,
   nbool               onlyContainers,
   pBuf_T              subordinates);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSExtSyncList
  (context : NWDSContextHandle;
   objectName : pnstr8;
   className : pnstr8;
   subordinateName : pnstr8;
   iterationHandle : pnint32;
   timeStamp : pTimeStamp_T;
   onlyContainers : nbool;
   subordinates : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object whose immediate subordinate objects are to be listed.

className

(IN) Points to a class name to be used as a filter (can contain wildcards).

subordinateName

(IN) Points to an object name to be used as a filter (can contain wildcards).

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSExtSyncList**. This should be set to NO_MORE_ITERATIONS initially.

timeStamp

(IN) Points to an object-modification time to be used as a filter (can be NULL).

onlyContainers

(IN) Specifies whether the results should include only container objects: TRUE=only container objects; FALSE=other objects.

subordinates

(OUT) Points to a Buf_T containing a list of subordinate objects matching the filters.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The name specified by *className*'s filter is the name of an object class, such as User, Computer, or Server. It can be a specific name or a string containing wildcards. A wildcard can be a zero-length string, or a string containing asterisks (*):

"" or "*" specifies all class names.

"U*" specifies all class names beginning with "U".

The value given for *subordinateName*'s filter can be one of the following:

The left-most name of an object, such as Adam or Graphics Printer.

A string with asterisks (*), such as A* or Gr*.

A zero length string (""), which means any name is valid.

The following examples show how to use wildcards for untyped names:

c* Any object whose left-most name begins with a "c" character.

M*y Any object beginning with "M" and ending with"y" such as Mary.

If the wildcard name specified for *subordinateName* includes a type, such as "CN," the name must include the equals (=) sign. The following examples show how to use wildcards for typed names:

cn=* Any object whose left-most name is a common name.

cn=c* Any object whose left-most name is a common name and begin with "c."

o*=* Any object whose left-most name is of an attribute type beginning with an "o," such as O or OU.

o*=c* Any object whose left-most name is of an attribute type beginning with an "o," and whose name begins with "c."

timeStamp's filter restricts the result to objects having modification times greater than or equal to the time specified in *timeStamp*.

When filling out TimeStamp_T, set *eventID* to zero, *replicaNum* to zero, and *wholeSeconds* to the appropriate value.

iterationHandle controls retrieval of search results larger than the result buffer pointed to by *subordinates*.

Before the initial call to **NWDSExtSyncList**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

If, when **NWDSExtSyncList** returns from its initial call, the result buffer holds the complete results, the location pointed to by *iterationHandle* is NO_MORE_ITERATIONS. If the iteration handle is not NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSExtSyncList** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be NO_MORE_ITERATIONS.

To end the List operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_SEARCH to free memory and states associated with the List operation.

onlyContainers specifies whether the results should be restricted to include information for container objects only. If *onlyContainers* is FALSE (0), the result contains information for objects of all object types. If any other value is given, only information for container objects is returned.

Allocate the result buffer pointed to by subordinates by calling **NWDSAllocBuf**. The result buffer does not need to be initialized because it is a result buffer.

The contents of the result buffer pointed to by *subordinates* is overwritten with each subsequent call to **NWDSExtSyncList**. Remove the contents from the result buffer before each subsequent call to **NWDSExtSyncList**.

The results of **NWDSExtSyncList** are not ordered and might not be in alphabetical order.

For more information, see Retrieving Results from NDS Output Buffers.

NOTE: On large networks, iterative processes, like **NWDSExtSyncList** , might take a lot of time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. These processes can be interrupted or aborted using **NWDSCloseIteration**.

Developers should use **NWDSCloseIteration** to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSList**, **NWDSListByClassAndName**, **NWDSListContainers**

NWDSExtSyncRead

Reads values from one or more of an NDS object's attributes allowing for restrictions on the attributes modification time

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1, 4.11

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSExtSyncRead (
   NWDSContextHandle   context,
   pnstr8              objectName,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint32             iterationHandle,
   pTimeStamp_T        timeStamp,
   pBuf_T              objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSExtSyncRead
  (context : NWDSContextHandle;
   objectName : pnstr8;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint32;
   timeStamp : pTimeStamp_T;
   objectInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object whose attributes are to be read.

infoType

(IN) Specifies the type of information desired.

allAttrs

(IN) Specifies the scope of the request.

attrNames

(IN) Points to a request buffer containing the attribute names for which information is to be returned.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSExtSyncRead**. This should be set initially to NO_MORE_ITERATIONS.

timeStamp

(IN) Points to an object-modification time to be used as a filter.

objectInfo

(OUT) Points to a result buffer that receives the attribute names or names and values.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The name specified by *objectName* is relative to the current name context in the NDS context specified by *context*.

infoType, *allAttrs*, *attrNames*, and *timeStamp* indicate what attribute information is requested.

infoType specifies whether both attribute names and attribute values are requested:

0 DS_ATTRIBUTE_NAME defines attribute names only.
1 DS_ATTRIBUTE_VALUES defines both attribute names and attribute values.

If *allAttrs* is TRUE, information about all attributes associated with the object is requested and *attrNames* is ignored (in which case, assign a NULL pointer to *attrNames*). If *allAttrs* is FALSE, only the attributes specified by the result buffer pointed to by *attrNames* are requested.

If *allAttrs* is FALSE and *attrNames* is NULL, no attribute information is returned. *infoType* is not meaningful. In this case, the return value of **NWDSExtSyncRead** can determine whether the specified object exists (verifying the objects distinguished name), or whether access to the object is allowed.

The request buffer pointed to by *attrNames* explicitly specifies the attributes whose information is to be returned. For more information, see Reading Attributes of NDS Objects.

The timestamp pointed to by *timeStamp* is used to exclude attributes that have not been modified since a certain time. The timestamp filter limits the attribute list to be those attributes having modification times greater than or equal to the specified time.

When filling out TimeStamp_T, set *eventID* to zero, *replicaNum* to zero, and *wholeSeconds* to the appropriate value.

On return, the result buffer pointed to by *objectInfo* contains the requested information. This result buffer is allocated by calling **NWDSAllocBuf**. It is not initialized since it is a result buffer.

This result buffer either contains a list of attribute names or a sequence of attribute-name and attribute-value sets. The type of information returned depends on *infoType*. For more information, see Retrieving Results from NDS Output Buffers.

If *allAttrs* is set to DS_ATTRIBUTE_VALUES, specifying the Read operation should return both attribute names and values, you cannot remove only names from the result buffer. You must remove the information in the correct order of attribute name first, then all of the values associated with the attribute. Then you remove the next attribute name and its values. Otherwise, **NWDSGetAttrName** will return erroneous information.

iterationHandle controls retrieval of search results larger than the result buffer pointed to by *objectInfo*.

Before the initial call to **NWDSExtSyncRead**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

If, when **NWDSExtSyncRead** returns from its initial call, the result buffer holds the complete results, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSExtSyncRead** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

To end the Read operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_READ to free memory and states associated with the Read operation.

The level of granularity for partial results is an individual value of an attribute. If an attribute is multivalued and its values are split across two or more **NWDSExtSyncRead** results, the attribute name is repeated in each result.

The results of **NWDSExtSyncRead** are not ordered and might not be in alphabetical order.

NWDSExtSyncRead can be useful for detecting changes in an object's attributes. However, **NWDSExtSyncRead** does not return information about attributes that have been deleted or for which your attribute privileges have changed.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRead, **NWDSReadObjectInfo**

NWDSExtSyncSearch

Searches a region of the NDS for objects satisfying a set of specified requirements, including modification time

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1, 4.11

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSExtSyncSearch (
   NWDSContextHandle   context,
   pnstr8              baseObjectName,
   nint                scope,
   nbool8              searchAliases,
   pBuf_T              filter
   pTimeStamp_T        timeStamp,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint32             iterationHandle,
   nint32              countObjectsToSearch,
   pnint32             countObjectsSearched,
   pBuf_T              objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSExtSyncSearch
  (context : NWDSContextHandle;
   baseObjectName : pnstr8;
   scope : nint;
   searchAliases : nbool8;
   filter : pBuf_T;
   timeStamp : pTimeStamp_T;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint32;
   countObjectsToSearch : nint32;
   countObjectsSearched : pnint32;
   objectInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

baseObjectName

(IN) Points to the name of a subtree root to be searched.

scope

(IN) Specifies the depth of the search.

searchAliases

(IN) Specifies whether to dereference aliases in the search subtree.

filter

(IN) Points to a Buf_T containing a search filter. This parameter must be specified (cannot be NULL).

timeStamp

(IN) Points to an object-modification time to further restrict the filter provided by *filter*. This parameter must be specified (cannot be NULL).

infoType

(IN) Specifies the type of information to be returned.

allAttrs

(IN) Specifies the scope of the request: TRUE=information concerning all attributes is requested; FALSE=only attributes named in *attrNames* is requested.

attrNames

(IN) Points to a Buf_T containing the attribute names for which information is to be returned.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSExtSyncSearch**. This should be set to NO_MORE_ITERATIONS initially.

countObjectsToSearch

(IN) Specifies the number of objects for the server to search before the server returns to the client.

countObjectsSearched

(OUT) Points to the number of objects searched by the server.

objectInfo

(OUT) Points to a Buf_T containing the names of the objects along with any requested attribute values satisfying the search.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSExtSyncSearch succeeds if the base object specified by *baseObjectName* is located, regardless of whether there are any subordinates to return.

baseObjectName identifies the object (or possibly the root) from which the search is relative. If the string is a zero-length string (""), the current name context specified in *context* is selected as the base object.

Aliases are dereferenced while locating the base object unless the context flag associated with DCV_DEREF_ALIASES is not set.

Aliases among the subordinates of the base object are dereferenced during the search unless *searchAliases* is FALSE. If *searchAliases* is TRUE, the search continues in the subtree of the aliases object.

scope takes one of three possible values:

0 DS_SEARCH_ENTRY indicates the search applies only to the base object.
1 DS_SEARCH_SUBORDINATES indicates the search applies only to the immediate subordinates of the base object.
2 DS_SEARCH_SUBTREE indicates the search applies to the base object and all its subordinates.

filter eliminates objects not of interest to the application. Information is returned only on objects that satisfy the filter. This filter is created by calling **NWDSAllocFilter**, **NWDSAddFilterToken**, and **NWDSPutFilter**.

When filling out TimeStamp_T, set *eventID* to zero, *replicaNum* to zero, and *wholeSeconds* to the appropriate value.

infoType, *allAttrs*, and *attrNames* indicate what attribute information is requested.

If *allAttrs* is TRUE, information about all attributes associated with the object is requested and *attrNames* is ignored (in which case, *attrNames* can be NULL). If *allAttrs* is FALSE, only the attributes specified by *attrNames* are requested.

If *allAttrs* is FALSE and *attrNames* is NULL, no attribute information is returned. *infoType* is not meaningful. In this case, the return value of **NWDSExtSyncSearch** simply determines whether the object specified by *baseObjectName* exists, or whether access to the object is allowed.

The request buffer pointed to by *attrNames* is used to explicitly specify the names of the attributes whose information is to be returned. For more information, see Preparing NDS Output Buffers.

On return, the buffer pointed to by *objectInfo* contains the information for objects matching the search criteria, along with the requested attribute information. For more information, see Retrieving Results from NDS Output Buffers.

You must retrieve all information from the buffer even if you do not plan to use it.

iterationHandle controls retrieval of search results larger than the result buffer pointed to by *objectInfo*.

Before the initial call to **NWDSExtSyncSearch**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

If, when **NWDSExtSyncSearch** returns from its initial call, the result buffer holds the complete results, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSExtSyncSearch** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

To end the Search operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_SEARCH to free memory and states associated with the Search operation.

The level of granularity for partial results is an individual attribute value. If the attribute is multivalued and its values are split across two or more calls to **NWDSExtSyncSearch**, the current object name, object info, and attribute name repeated the subsequent result buffer.

NOTE: Currently, because of aliasing, searching a subtree can result 1) in duplicate entries or 2) in an infinite loop.

NWDSExtSyncSearch can be useful for detecting changes in objects matching a search direction. However, **NWDSExtSyncSearch** does not return information about objects that have been deleted or for which your privileges have changed.

NOTE: On large networks, iterative processes, like **NWDSExtSyncSearch**, might take a lot of time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. These processes can be interrupted or aborted using **NWDSCloseIteration**.

Developers should use **NWDSCloseIteration** to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAddFilterToken, **NWDSAllocFilter**, **NWDSCloseIteration**, **NWDSFreeFilter**, **NWDSPutFilter**, **NWDSSearch**

NWDSFreeBuf

Frees a buffer allocated by the **NWDSAllocBuf** function
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSFreeBuf (
   pBuf_T   buf);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSFreeBuf
   (buf : pBuf_T
) : NWDSCCODE;
```

Parameters

buf

(IN) Points to the buffer to be freed.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0xFEB5	ERR_NULL_POINTER

Remarks

All buffers allocated by calling **NWDSAllocBuf** should be freed once they are no longer needed by the client. Doing so frees up memory for the client.

If the *buf* parameter is passed NULL, **NWDSFreeBuf** will return ERR_NULL_POINTER.

NCP Calls

None

See Also

NWDSAllocBuf, **NWDSInitBuf**

NWDSFreeContext

Frees a previously allocated NDS context

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdc.h>

NWDSCCODE N_API NWDSFreeContext (
   NWDSContextHandle   context);
```

Pascal Syntax

```
#include <nwdsdc.inc>

Function NWDSFreeContext
  (context : NWDSContextHandle
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context to be freed.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

All NDS contexts created by **NWDSCreateContextHandle** or **NWDSDuplicateContext** should be freed when the client is no longer using them. Doing so frees memory for the client.

NCP Calls

None

See Also

NWDSCreateContextHandle, **NWDSGetContext**, **NWDSSetContext**

NWDSFreeFilter

Frees the area allocated to a search filter expression tree
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

void N_API NWDSFreeFilter (
   pFilter_Cursor_T       cur,
   void (N_FAR N_CDECL  *freeVal)(nuint32 syntax, nptr val);
```

Pascal Syntax

```
#include <nwdsfilt.inc>

Function NWDSFreeFilter
  (cur : pFilter_Cursor_T;
   freeVal : FreeValProc
);
```

Parameters

cur

(IN) Points to the filter to be freed allocated from **NWDSAllocFilter**.

freeVal

(IN) Specifies the function to be used to free nodes in the filter expression tree; can be NULL.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Normally, the expression tree is freed by **NWDSPutFilter** when the tree is stored in the request buffer. If the tree is not used, it should be freed by calling **NWDSFreeFilter**.

The function specified by *freeVal* must accept two parameters.

Do not call **NWDSFreeFilter** after calling **NWDSPutFilter**, even if **NWDSPutFilter** returns an error.

NCP Calls

None

See Also

NWDSAddFilterToken, **NWDSAllocFilter**, **NWDSDelFilterToken**, **NWDSPutFilter**

NWDSGenerateObjectKeyPair

Creates or changes a public/private key pair for a specified object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsasa.h>

NWDSCCODE N_API NWDSGenerateObjectKeyPair (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              objectPassword,
   nflag32             optionsFlag);
```

Pascal Syntax

```
#include <nwdsasa.inc>

Function NWDSGenerateObjectKeyPair
  (contextHandle : NWDSContextHandle;
   objectName : pnstr8;
   objectPassword : pnstr8;
   optionsFlag : nflag32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object to update.

objectPassword

(IN) Points to the object password in ASCII text format.

optionsFlag

(IN) Is reserved (pass in zero).

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

If no password is desired, *objectPassword* should point to a zero-length string ("").

If an application has a local copy of any password value, the value should be erased as soon as possible to prevent compromising the security of the password.

An object must have rights to modify an objects attributes before the **NWDSGenerateObjectKeyPair** function will succeed.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSChangeObjectPassword

NWDSGetAttrCount

Returns the number of attributes whose information is stored in a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetAttrCount (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            attrCount);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetAttrCount
  (context : NWDSContextHandle;
   buf : pBuf_T;
   attrCount : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to Buf_T.

attrCount

(OUT) Points to the number of attributes in the result buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetAttrCount should be the first "Get" operation performed following a Read operation (such as **NWDSRead**, **NWDSReadAttrDef**, or **NWDSSearch**).

After the attribute count has been determined, the attribute names can be retrieved from the buffer by calling **NWDSGetAttrName** or **NWDSGetAttrDef**. Attribute values are retrieved using a combination of calls to **NWDSComputeAttrValSize** and **NWDSGetAttrVal**.

buf points to a Buf_T filled in by a previous call to a NDS function, such as **NWDSRead**.

NCP Calls

None

See Also

NWDSGetAttrDef, **NWDSGetAttrName**, **NWDSRead**, **NWDSReadAttrDef**

NWDSGetAttrDef

Returns the next NDS Schema attribute definition from a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetAttrDef (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              attrName,
   pAttr_Info_T        attrInfo);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetAttrDef
  (context : NWDSContextHandle;
   buf : pBuf_T;
   attrName : pnstr8;
   attrInfo : pAttr_Info_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

attrName

(OUT) Points to the name of the attribute definition at the current position in the result buffer.

attrInfo

(OUT) Points to additional information about the attribute definition.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetAttrDef is used to retrieve attribute information from a result buffer filled in by **NWDSReadAttrDef**. For more information, see Retrieving Results from NDS Output Buffers.

You must allocate space for the attribute name pointed to by *attrName*. The size of the allocated memory is ((MAX_SCHEMA_NAME_CHARS)+1)*sizeof(character size) where character size is for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

If **NWDSReadAttrDef** is called with *infoType* set to DS_ATTR_DEF_NAMES (instead of DS_ATTR_DEFS), its output buffer will contain only names of the attributes. In this case, **NWDSGetAttrDef** ignores *attrInfo*, so *attrInfo* can be NULL.

You must allocate memory (sizeof(Attr_Info_T)) to receive the additional attribute-definition information.

NCP Calls

None

See Also

NWDSGetAttrCount, **NWDSReadAttrDef**

NWDSGetAttrName

Retrieves the name of the attribute whose information is stored at the current position in a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetAttrName (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              attrName,
   pnuint32            attrValCount,
   pnuint32            syntaxID);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetAttrName
  (context : NWDSContextHandle;
   buf : pBuf_T;
   attrName : pnstr8;
   attrValCount : pnuint32;
   syntaxID : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

attrName

(OUT) Points to the attribute name whose information is stored at the current position in the result buffer.

attrValCount

(OUT) Points to the number of attribute values following the attribute name in the result buffer. (Multivalued attributes can have more than one value.)

syntaxID

(OUT) Points to the syntax ID identifying the syntax type of the attribute returned in *attrName*.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetAttrName is used to retrieve attribute information from a result buffer filled in by **NWDSRead**, **NWDSSearch**, or **NWDSList**.

You must allocate space for the attribute name. The size of the allocated memory is ((MAX_SCHEMA_NAME_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

The location pointed to by *attrValCount* is set to specify the number of attribute values associated with the current attribute in the result buffer. If no values are associated with the current attribute, the number will be zero. If the current attribute is a single-valued attribute, the number will be one. If the current attribute is a multi-valued attribute, the number can be zero or more.

The location pointed to by *syntaxID* receives a value identifying the syntax type of the attribute returned in *attrName*. This ID is passed as a parameter to subsequent calls to **NWDSComputeAttrValSize** and **NWDSGetAttrVal**. The syntax types (such as SYN_CI_STRING) are enumerated in NWDSDEFS.H.

If the function filling in the result buffer was called specifying that the results contain only names, **NWDSGetAttrName** will not place a value into the locations pointed to by *attrValCount* and *syntaxID*. In this case, *attrValCount* and *syntaxID* can be NULL.

For more information, see Reading Attributes of NDS Objects.

NCP Calls

None

See Also

NWDSGetAttrCount, **NWDSRead**, **NWDSSearch**, **NWDSReadAttrDef**

NWDSGetAttrVal

Returns the next attribute value in a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetAttrVal
   (NWDSContextHandle   context,
    pBuf_T              buf,
    nuint32             syntaxID,
    nptr                attrVal);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetAttrVal
   (context : NWDSContextHandle;
    buf : pBuf_T;
    syntaxID : nuint32;
    attrVal : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

syntaxID

(IN) Specifies the syntax of the attribute value.

attrVal

(OUT) Points to the attribute value at the current buffer position.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetAttrVal is used to retrieve attribute values from a result buffer filled in by functions such as **NWDSList**, **NWDSRead**, or **NWDSSearch**.

syntaxID is returned by a previous call to **NWDSGetAttrName**. *syntaxID* indicates to **NWDSGetAttrVal** how to translate the attribute value into a data structure. The structure of the data returned in *attrVal* depends on the value of *syntaxID*.

The syntax types (such as SYN_CI_STRING) are enumerated in NWDSDEFS.H. Attribute syntaxes and their corresponding data structures are listed in NDS Attribute Syntax Definitions.

If *attrVal* equals NULL, the value is skipped; this is useful for simply counting attribute values.

You must allocate memory for the attribute value and set *attrVal* to point to that memory. The memory must be a contiguous block of memory whose size is determined by calling **NWDSComputeAttrValSize**.

The memory pointed to by *attrVal* should be dynamically allocated memory since the size of the memory needed to store the attribute values can be different even when the values are associated with the same attribute.

See Reading Attributes of NDS Objects for the steps to remove attribute values from a result buffer.

NCP Calls

None

NWDSGetBinderyContext

Returns the setting of the Bindery context set on the server identified by *conn*

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSGetBinderyContext (
   NWDSContextHandle   context,
   NWCONN_HANDLE       conn,
   pnstr8              BinderyEmulationContext);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSGetBinderyContext
  (context : NWDSContextHandle;
   conn : NWCONN_HANDLE;
   BinderyEmulationContext : pnuint8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

conn

(IN) Specifies the NetWare server connection handle.

binderyEmulationContext

(OUT) Points to a Bindery context string.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x89FE	BAD_PACKET
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

conn is the connection handle to the server in which you are interested.

binderyEmulationContext must have sufficient space allocated to receive the value.

Bindery connections to NetWare 4.x servers are communicating with the server in Bindery mode. The Bindery context specifies a location in NDS where a Bindery connection is allowed to see objects in NDS. A Bindery connection can see objects only at the level of the tree defined by the server's Bindery context.

Bindery context is set on NetWare 4.x servers by using the **SET BINDERY CONTEXT** command at the server console.

NCP Calls

0x2222 104 04	Return Bindery Context

See Also

NWDSAuditGetObjectID

NWDSGetClassDef

Retrieves an object-class definition from a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetClassDef (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              className,
   pClass_Info_T       classInfo);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetClassDef
  (context : NWDSContextHandle;
   buf : pBuf_T;
   className : pnstr8;
   classInfo : pClass_Info_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

className

(OUT) Points to the name of the object-class definition at the current position in the buffer.

classInfo

(OUT) Points to the initial portion of the object-class definition.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetClassDef is used to retrieve class definitions from a result buffer filled in by **NWDSReadClassDef**.

className points to the name of the current class in the buffer. You must allocate space for the class name. The size of the allocated memory is ((MAX_SCHEMA_NAME_CHARS)+1)*sizeof(characters size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

If **NWDSReadClassDef** is called with *infoType* set to DS_CLASS_DEF_NAMES, *classInfo* of **NWDSGetClassDef** is ignored and can be NULL.

The complete steps for retrieving class information from a result buffer are listed in the reference for **NWDSReadClassDef**.

NCP Calls

None

See Also

NWDSGetClassDefCount, **NWDSGetClassItem**, **NWDSGetClassItemCount**, **NWDSReadClassDef**

NWDSGetClassDefCount

Returns the number of object-class definitions stored in a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetClassDefCount (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            classDefCount);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetClassDefCount
  (context : NWDSContextHandle;
   buf : pBuf_T;
   classDefCount : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

classDefCount

(OUT) Points to the number of object-class definitions stored in the buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetClassDefCount is used to determine the number of object-class definitions stored in a result buffer filled by **NWDSReadClassDef**.

NWDSGetClassDefCount must be the first function called when reading a result buffer containing a group of object-class definitions.

The complete steps for retrieving class information from a result buffer are listed in the reference for **NWDSReadClassDef**.

NCP Calls

None

See Also

NWDSGetClassDef

NWDSGetClassItem

Returns the name of the next object class item stored in a result buffer
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetClassItem (
   (NWDSContextHandle   context,
    pBuf_T              buf,
    pnstr8              itemName);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetClassItem
   (context : NWDSContextHandle;
    buf : pBuf_T;
    itemName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

itemName

(OUT) Points to the name of the item (attribute or class) at the current position in the result buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

buf points to a Buf_T filled in by **NWDSReadClassDef**.

itemName points to the name of either an attribute or a class. The item is a member of one of the five class-definition-item lists:

1. Super Class Names
2. Containment Class Names
3. Naming Attribute Names
4. Mandatory Attribute Names
5. Optional Attribute Names

The user must allocate space for the class item name pointed to by *itemName*. The size of the allocated memory is ((MAX_SCHEMA_NAME_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

Before retrieving the class items from a class-definition-item list, determine the number of items in the list by calling **NWDSGetClassItemCount**. Then retrieve the items associated with the list by repeatedly calling **NWDSGetClassItem** once for each item in the list. Then determine the number of items in the next list by calling **NWDSGetClassItemCount**, and retrieve the values for the list by calling **NWDSGetClassItem**, and so on until you have retrieved all of the information from all of the lists.

NOTE: You must retrieve the information from the class-definition-item lists in the order shown above.

For the complete steps for reading class-definition information, see Reading a Class Definition.

NCP Calls

None

See Also

NWDSGetClassDef, **NWDSGetClassItemCount**, **NWDSListContainableClasses**, **NWDSReadClassDef**

NWDSGetClassItemCount

Returns the number of object class definition items associated with a result buffer's current object class definition list in a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetClassItemCount (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            itemCount);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetClassItemCount
  (context : NWDSContextHandle;
   buf : pBuf_T;
   itemCount : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

itemCount

(OUT) Points to the number of object-class definition items associated with the result buffer's current class-definition-item list.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

buf points to Buf_T filled in by **NWDSReadClassDef**.

itemCount points to the number of object-class-definition items that are associated with the current class-definition-item list. There are five class-definition item lists; these lists are stored in the buffer in the following order:

1. Super Class Names
2. Containment Class Names
3. Naming Attribute Names
4. Mandatory Attribute Names
5. Optional Attribute Names

The first two lists contain the names of classes. The remaining lists contain the names of attributes.

Before retrieving class items from a class-definition-item list, determine the number of items in the list by calling **NWDSGetClassItemCount**. Retrieve the items associated with the list by calling **NWDSGetClassItem** once for each item in the list. Then determine the number of items in the next list by calling **NWDSGetClassItemCount**, and retrieve the values for the list by calling **NWDSGetClassItem**, until you have retrieved all of the information from all lists.

For the complete steps for reading object-class-definition information, see Reading a Class Definition.

NCP Calls

None

See Also

NWDSGetClassDef, **NWDSGetClassItem**, **NWDSListContainableClasses**, **NWDSReadClassDef**

NWDSGetConnectionInfo (obsolete 6/96)

Returns connection information relating to the new data included in the connection table but is now obsolete. Call **NWCCGetConnInfo** instead.

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdstype.h>
#include <nwndscon.h>

NWCCODE N_API NWDSGetConnectionInfo
  (NWCONN_HANDLE      conn,
   nuint8 N_FAR *     connStatus,
   nuint8 N_FAR *     connType,
   nuint8 N_FAR *     serverFlags,
   nuint8 N_FAR *     serverName,
   nuint8 N_FAR *     transType,
   nuint32 N_FAR *    transLen,
   nuint8 N_FAR *     transBuf,
   nuint16 N_FAR *    distance,
   nuint16 N_FAR *    maxPacketSize);
```

Pascal Syntax

```
Function NWDSGetConnectionInfo
  (conn : NWCONN_HANDLE;
   connStatus : pnuint8;
   connType : pnuint8;
   serverFlags : pnuint8;
   serverName : pnuint8;
   transType : pnuint8;
   transLen : pnuint32;
   transBuf : pnuint8;
   distance : pnuint16;
   maxPacketSize : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

connStatus

(OUT) Points to the lower byte of *connectionFlags* in the connection table.

connType

(OUT) Points to NDSType in the connection table.

serverFlags

(OUT) Points to *serverFlags* in the connection table. It is used to find SFT III™ and Packet Burst™ status for the connection.

serverName

(OUT) Points to the ASCII name of the server associated with the connection.

transType

(OUT) Points to the transport protocol for the connection (IPX* protocol).

transLen

(OUT) Points to the transport buffer length (IPX = 12).

transBuf

(OUT) Points to the transport dependent address information (12 bytes for IPX).

distance

(OUT) Points to the distance to the server in time relative to other connections.

maxPacketSize

(OUT) Points to the number of bytes this connection can send or receive in one packet.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8836	INVALID_PARAMETER

Remarks

NWDSGetConnectionInfo (obsolete 6/96) handles NULL parameters; therefore, not all parameters need be declared and allocated.

To call **NWDSGetConnectionInfo (obsolete 6/96)** in DOS or in Windows, VLMs must be running. NETX does not support **NWDSGetConnectionInfo (6/96)** and will return an error if VLMs are not running.

connType returns the following values:

```
0x01   NDS Connection
0x02   Connection to NetWare server is licensed
0x04   NDS Authenticate.
```

serverFlags' bit definitions follow:

```
0x01   Packet Burst Available
0x10   SFT III Changing Servers
0x20   Packet Burst Session Reset
0x40   Need Max IO
0x80   SFT III Server Has Change.
```

NCP Calls

None

NWDSGetConnectionSlot (obsolete 6/96)

Opens a connection given the specified type but is now obsolete. Call **NWCCOpenConnByAddr** followed by a call to **NWCCLicenseConn** instead.

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdstype.h>
#include <nwndscon.h>

NWCCODE N_API NWDSGetConnectionSlot
  (nuint8                  connType,
   nuint8                  transType,
   nuint32                 transLen,
   nuint8 N_FAR *          transBuf,
   NWCONN_HANDLE N_FAR *   conn);
```

Pascal Syntax

```
Function NWDSGetConnectionSlot
  (connType : nuint8;
   transType : nuint8;
   transLen : nuint32;
   transBuf : pnuint8;
   Var conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

connType

(IN) Specifies either a hard or soft allocation.

transType

(IN) Specifies the transport protocol for this connection; for example, IPX=1.

transLen

(IN) Specifies the length of the transport buffer; for example, IPX=12.

transBuf

(IN) Points to the transport dependent address information (12 bytes for IPX).

conn

(OUT) Points to the connection handle of the requested connection.

Return Values

These are common return values; see Return Values for more information.

0x000 0	SUCCESSFUL
0x005 F	Error Interrupt (Workstation Error)
0x800 1	Lan Err Out Of Resource (LSL™ Error)
0x880 1	INVALID_CONNECTION
0x883 F	CONNECTION_TABLE_FULL
0x884 1	BAD_TRAN_TYPE
0x89F F	NO_RESPONSE_FROM_SERVER
0x900 2	Ipx No Route (IPX Error)

Remarks

To call **NWDSGetConnectionSlot (obsolete 6/96)** in DOS or in Windows, VLMs must be running. NETX does not support **NWDSGetConnectionSlot (obsolete 6/96)** and will return an error if VLMs are not running.

NWDSGetConnectionSlot (obsolete 6/96) will return a licensed connection to the NetWare server if the connection is authenticated. Call **NWDSUnlockConnection** to unlicense the connection or place it on the Least Recently Used (LRU) list to be cached.

connType increments the global use count and increments the task use count for the given connection. If the task goes away without freeing the connection slot, the global use count is decremented by the number in the task use count. This causes the connection to return to the state it was in before the process started.

connType can have the following values.

```
SYSTEM_LOCK        hard allocation
```

```
  TASK_LOCK          soft allocation
  TASK_DISCONNECT    no resource allocatio.
```

NCP Calls

None

NWDSGetContext

Returns the value of an NDS context variable

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdc.h>

NWDSCCODE N_API NWDSGetContext (
   NWDSContextHandle   context,
   nint                key,
   nptr                value);
```

Pascal Syntax

```
#include <nwdsdc.inc>

Function NWDSGetContext
  (context : NWDSContextHandle;
   key : nint;
   value : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the directory context to be queried.

key

(IN) Specifies the context variable to be retrieved.

value

(OUT) Points to the value of the context variable.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Applications do not have direct access to the NDS context variables. To determine the settings of the context variables, applications must call **NWDSGetContext** using a context key to identify for which variable information should be retrieved. See DSI Flags.

To call **NWDSGetContext** for multiple NDS identities, set the *key* parameter to DCK_TREE_NAME.

The **NWDSGetContext** function can be called using DCK_NAME_CACHE_DEPTH to query or set the depth of the name cache (how many names the cache will remember for the context handle). See Name Caching.

The *context* parameter usually defaults to the preferred tree name. Under NLM, the *context* parameter defaults to the local server tree name.

The context key is identified by the *key* parameter. The following keys are defined in NWDSDC.H:

1	DCK_FLAGS	nuint32	Bit definitions
2	DCK_CONFIDENCE	nuint32	Definitions: `0 DCV_LOW_CONF 1 DCV_MED_CONF 2 DCV_HIGH_CONF`
3	DCK_NAME_CONTEXT		Character string array
4	DCK_TRANSPORT_TYPE	nuint32[2]	Not currently in use
5	DCK_REFERRAL_SCOPE	nuint32	Definitions: `0 DCV_ANY_SCOPE 1 DCV_COUNTRY_SCOPE 2 DCV_ORGANIZATION_SCOPE 3 DCV_LOCAL_SCOPE`
8	DCK_LAST_CONNECTION		Returns NWCONN_HANDLE
11	DCK_TREE_NAME		Character string array of at most NW_MAX_TREE_NAME_LEN (includes NULL) ASCII or UNICODE characters

The *value* parameter should point to a variable of a type appropriate to receive the specified variable.

The flags associated with DCK_FLAGS are defined as follows:

0x001 L	$00000001	DCV_DEREF_ALIASES
0x002 L	$00000002	DCV_XLATE_STRINGS
0x004 L	$00000004	DCV_TYPELESS_NAMES
0x008 L	$00000008	DCV_ASYNC_MODE
0x010 L	$00000010	DCV_CANONICALIZE_NAMES
0x040 L	$00000040	DCV_DEREF_BASE_CLASS
0x080 L	$00000080	DCV_DISALLOW_REFERRALS

DCV_DEREF_ALIASES, DCV_XLATE_STRINGS, DCV_CANONICALIZE_NAMES, DCV_ANY_SCOPE, and DCV_LOW_CONF are set by default.

If the *key* parameter is DCK_CONFIDENCE, the value pointed to by the *value* parameter can be one of the following:

0	DCV_LOW_CONF
1	DCV_MED_CONF
2	DCV_HIGH_CONF

If the *key* parameter is DCK_NAME_CONTEXT, the *value* parameter points to a buffer containing the name context. You must allocate space for this buffer. The size of the memory needed to store the name context is ((MAX_DN_CHARS)+1*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

If *key* is DCK_REFERRAL_SCOPE, *value* can be one of the following:

0	DCV_ANY_SCOPE
1	DCV_COUNTRY_SCOPE
2	DCV_ORGANIZATION_SCOPE
3	DCV_LOCAL_SCOPE

NDS context variables can be changed by calling **NWDSSetContext**.

NCP Calls

None

See Also

NWDSCreateContextHandle, **NWDSSetContext**

NWDSGetCountByClassAndName

Counts the immediate subordinates of an NDS object, restricting the count to be of objects of a specified object class and/or with a specific name

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSGetCountByClassAndName (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              className,
   pnstr8              subordinateName,
   pnuint32            count);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSGetCountByClassAndName
  (context : NWDSContextHandle;
   objectName : pnstr8;
   className : pnstr8;
   subordinateName : pnstr8;
   count : pnint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the object name whose subordinates are to be counted.

className

(IN) Points to the class name to be used as a filter when determining which objects should be counted.

subordinateName

(IN) Points to a name to be used as a filter when determining which objects should be counted.

count

(OUT) Points to the count of subordinate objects matching the filters.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetCountByClassAndName is similar to **NWDSListByClassAndName** except no information, other than a count of objects, is returned by **NWDSGetCountByClassAndName**.

The location of the subordinate object(s) in the NDS tree is immediately subordinate to the object specified by *objectName*. It is not relative to the current name context in the NDS context specified by *context*.

The relationship between *className* and *subordinateName* is an "AND" relationship.

When *className* and *subordinateName* are provided, a count of immediate subordinate objects restricted by both filters is returned.

When *className* is NULL and *subordinateName* is NULL, the count of all immediate subordinates is returned.

When *className* is provided and *subordinateName* is NULL, the count of immediate subordinates restricted only by *className*'s filter is returned.

When *subordinateName* is provided and *className* is NULL, the count of immediate subordinates restricted only by *subordinateName* is returned.

The following examples show how to use wildcards for untyped names:

c*	Any object whose left-most name begins with a "c" character.
M*y	Any object beginning with "M" and ending with"y" such as Mary.

If the wildcard name specified for *subordinateName* includes a type, such as "CN," the name must include the equals (=) sign. The following examples show how to use wildcards for typed names:

cn=*	Any object whose left-most name is a common name.
cn=c*	Any object whose left-most name is a common name and begin with "c."
o*=*	Any object whose naming attribute is of a type beginning with an "o," such as O or OU.
o*=c*	Any object whose left-most name is of a type beginning with an "o," and whose name begins with "c."

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSListByClassAndName

NWDSGetDefNameContext

Retrieves the default name context for a specified tree

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSGetDefNameContext (
   NWDSContextHandle   context,
   nuint               nameContextLen
   pnstr8              nameContext);
```

Pascal Syntax

```
#include <nwdsconn.inc>

Function NWDSGetDefNameContext
  (context : NWDSContextHandle;
   nameContextLen : nuint;
   nameContext : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

nameContextLen

(IN) Specifies the length (in bytes) of the *nameContext* buffer.

nameContext

(OUT) Points to the buffer in which to place the default name context value.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

This function gets the default name context for the tree specified in the context (or if the tree name isn't set, the preferred tree name). This call differs from **NWGetDefaultNameContext** in that this call has an added parameter, *context*, and operates on a per tree basis. Also, this function call can return the name context in Unicode while **NWGetDefaultNameContext** could only return the data in local code page format.

The default name context for the preferred tree can be set by the DEFAULT NAME CONTEXT configuration parameter, or by calling either **NWSetDefaultNameContext** or **NWDSSetDefNameContext**. The default name context for another tree (different from the preferred tree) can be set by calling **NWDSSetDefNameContext**.

The default name context can be from 0 to 257 bytes long for local code page strings (including the NULL), or 0 to 514 bytes long for Unicode strings (including the 2 bytes for NULL). If the *nameContext* buffer is too small to contain the value, an error is returned and no data is copied.

If the treename is not set in the context, the preferred tree will be used. For requesters that do not support multiple trees, if the treename is specified (not -NULL string) and if the treename is different than the preferred tree, an error will be returned.

NCP Calls

None

See Also

NWGetDefaultNameContext, **NWSetDefaultNameContext**, **NWDSSetDefNameContext**

NWDSGetDSIInfo

Returns DSI object information not stored in the attributes of an object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_GLOBAL_LIBRARY (NWDSCCODE) NWDSGetDSIInfo (
   NWDSContextHandle   context,
   nptr                buf,
   nuint32             bufLen,
   nuint32             infoFlag,
   nptr                data);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetDSIInfo (
   context : NWDSContextHandle;
   buf : nptr;
   bufLen : nuint32;
   infoFlag : nuint32;
   data : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the buffer returned from previously calling the **NWDSRead**, **NWDSReadObjectDSIInfo**, **NWDSList** and/or **NWDSSearch** functions.

bufLen

Specifies the length of the *buf* parameter.

infoFlag

Specifies the data element to be extracted from the buffer pointed to in the *buf* parameter.

data

Points to a buffer to receive the data element value.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The **NWDSReadObjectDSIInfo** function will return data regarding an NDS object. **NWDSGetDSIInfo** will extract the individual data elements from the reply buffer. The returned "data" is formatted according to the data type of the element referred to by the DSI Flags.

Object information can be useful to applications browsing the NDS tree.

NCP Calls

None

See Also

NWDSGetObjectNameAndInfo, **NWDSReadObjectDSIInfo**

NWDSGetDSVerInfo

Returns NDS version information

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSGetDSVerInfo (
   NWCONN_HANDLE   conn,
   pnuint32        dsVersion,
   pnuint32        rootMostEntryDepth,
   pnstr8          sapName,
   pnuint32        flags,
   punicode        treeName);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSGetDSVerInfo
  (conn : NWCONN_HANDLE;
   Var dsVersion : nuint32;
   Var rootMostEntryDepth : nuint32;
   sapName : pnstr8;
   Var flags : nuint32;
   treeName : punicode
) : NWDSCCODE;
```

Parameters

conn

(IN) Specifies the connection handle to an NDS server.

dsVersion

(OUT) Specifies the DS.NLM build version.

rootMostEntryDepth

(OUT) Specifies the number of levels to the root-most object.

sapName

(OUT) Points to the tree name where the server is contained. The value is in the SAP form (ASCII characters set restricted by SAP).

flags

(OUT) Points to the DS.NLM flags.

(OUT) Points to the DS.NLM flags.

treeName

(OUT) Points to the "enabled" tree name (in the Unicode character set).

Return Values

0x0000	SUCCESSFUL
nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetDSVerInfo returns version information regarding the DS.NLM running on a specific server. Each return value is optional (that is, passing a NULL as the pointer disables the return of the information).

NCP Calls

None

See Also

NWDSGetNDSStatistics, **NWGetNWNetVersion**

NWDSGetEffectiveRights

Returns a summary of a subject's rights with respect to operations on a specified object or an attribute of an object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsacl.h>

NWDSCCODE N_API NWDSGetEffectiveRights (
   NWDSContextHandle   context,
   pnstr8              subjectName,
   pnstr8              objectName,
   pnstr8              attrName,
   pnuint32            privileges);
```

Pascal Syntax

```
#include <nwdsacl.inc>

Function NWDSGetEffectiveRights
  (context : NWDSContextHandle;
   subjectName : pnstr8;
   objectName : pnstr8;
   attrName : pnstr8;
   privileges : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

subjectName

(IN) Points to the name of the object to which the privileges are assigned.

objectName

(IN) Points to the name of the object to which access may be granted.

attrName

(IN) Points to the name of the attribute to which access may be granted.

privileges

(OUT) Points to the privileges assigned to *subjectName*.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

If the return value is ERROR_NO_SUCH_ENTRY, no privilege set exists for the specified subject/object pair, and the subject has no rights with respect to the object. It can also indicate the object does not exist.

If the object exists but the subject does not exist, **NWDSGetEffectiveRights** returns a value of SUCCESSFUL and *privileges* is set to NULL.

Access to information about objects stored in NDS is granted through access control lists (ACLs). The ACL is an attribute defined by the NDS Schema and regulates access to its associated object or attribute. The ACL can be read or modified by calling **NWDSRead** and **NWDSModifyObject**. Likewise, other access operations can be applied to the ACL.

The ACL grants access privileges to a specified object, called the subject, regarding the object the ACL protects. Optionally, privileges may be granted with respect to a specified attribute of the protected object.

A subject can inherit access to an object through various security equivalences. **NWDSGetEffectiveRights** provides a summary of all cases where a particular subject may receive access to a particular object. (The value for individual ACLs can be read or modified using the standard Access Services.)

The subject can be the name of the objects in NDS, or it can be one of the following "special" subjects:

```
Special Subjects
      [Creator]
      [Public]
      [Root]
      [Self]
```

The [Inheritance Mask] special subject cannot be used. **NWDSGetEffectiveRights** will return -601, ERR_NO_SUCH_ENTRY, when trying to get the inheritance mask for a container or user.

attrName specifies an attribute of the object for which the effective rights of the subject are requested. The attribute can also be one of the following "special" attribute names:

```
Special Attribute Names
      All Attributes Rights
      Entry Rights
      SMS Rights
```

The *privileges* parameter returns the effective privilege set for subject/object or subject/attribute pair.

All Attributes Rights:

0x001L	$00000001	DS_ATTR_COMPARE
0x002L	$00000002	DS_ATTR_READ
0x004L	$00000004	DS_ATTR_WRITE
0x008L	$00000008	DS_ATTR_SELF
0x010L	$00000010	DS_ATTR_SUPERVISOR

Entry Rights:

0x001L	$00000001	DS_ENTRY_BROWSE

0x002 L	$00000002	DS_ENTRY_ADD
0x004 L	$00000004	DS_ENTRY_DELETE
0x008 L	$00000008	DS_ENTRY_RENAME
0x010 L	$00000010	DS_ENTRY_SUPERVISOR

SMS Rights

0x001 L	$00000001	DS_SMS_SCAN
0x002 L	$00000002	DS_SMS_BACKUP
0x004 L	$00000004	DS_SMS_RESTORE
0x008 L	$00000008	DS_SMS_RENAME
0x010 L	$00000010	DS_SMS_DELETE
0x020 L	$00000020	DS_SMS_ADMIN

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

NWDSGetMonitoredConnection (obsolete 6/96)

Returns the connection handle of a monitored NDS connection but is now obsolete. Call **NWDSOpenMonitoredConn** instead.

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdstype.h>
#include <nwndscon.h>

NWCCODE N_API NWDSGetMonitoredConnection
  (NWCONN_HANDLE N_FAR *   conn);
```

Pascal Syntax

```
Function NWDSGetMonitoredConnection
  (Var conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(OUT) Points to the connection handle.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x886F	OBJECT_NOT_FOUND

Remarks

A monitored connection is set only if **NWDSLogin** has been called.

NWDSLogin creates a set of attributes in NDS of a NetWare server. The connection to the NetWare server must be maintained for the duration of the login session; otherwise, the attributes will be deleted when the connection is destroyed. **NWDSGetMonitoredConnection (obsolete 6/96)** is the place holder for the connection to the NetWare server having these attributes.

When **NWDSLogout** is called, the connection handle is set to zero (0).

A monitored connection does not exist if *conn* is zero (0).

NWDSGetMonitoredConnection (obsolete 6/96) does not guarantee a licensed connection to the NetWare server.

NCP Calls

None

NWDSGetMonitoredConnRef

Retrieves a monitored connection reference
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSGetMonitoredConnRef (
   NWDSContextHandle   context,
   pnuint32            connRef);
```

Pascal Syntax

```
#include <nwdsconn.inc>

Function NWDSGetMonitoredConnRef
  (context : NWDSContextHandle;
   Var connRef : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connRef

(OUT) Points to the connection reference.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

A monitored connection reference is set only if **NWDSLogin** has been called. For multiple tree support, the tree name specified in the context handle is used to specify which monitored connection reference to retrieve.

If the treename is not set in the context, the preferred tree will be used. For requesters that do not support multiple trees, if the treename is specified (not -NULL string) and if the treename is different than the preferred tree, an error will be returned.

To make use of the connection reference, a connection handle must be opened using the connection reference.

NCP Calls

None

See Also

NWDSOpenMonitoredConn, NWCCOpenConnByRef

NWDSGetNDSStatistics

Retrieves the NDS statistics

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSGetNDSStatistics (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   nuint               statsInfoLen,
   pNDSStatsInfo_T     statsInfo);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSGetNDSStatistics
  (context : NWDSContextHandle;
   serverName : pnstr8;
   statsInfoLen : nuint;
   Var statsInfo : NDSStatsInfo_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

serverName

(IN) Specifies the server to send the request to.

statsInfoLen

(OUT) Receives the length (in bytes) of the statsInfo structure.

statsInfo

(OUT) Points to a structure that contains statistical information for NDS relative to the local server described by *serverName*.

Return Values

0x0000	SUCCESSFUL

nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetNDSStatistics retrieves all of the statistics that can be reported by NDS.

NCP Calls

None

See Also

NWDSResetNDSStatistics,

NWDSGetObjectCount

Returns the number of objects whose information is stored in a result buffer
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetObjectCount (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            objectCount);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetObjectCount
  (context : NWDSContextHandle;
   buf : pBuf_T;
   objectCount : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

objectCount

(OUT) Points to the number of objects whose information is stored in the result buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetObjectCount must be the first function used to read a result buffer containing information about objects, such as result buffers filled in by **NWDSList**, **NWDSRead**, and **NWDSSearch**.

The full steps for retrieving object information from a result buffer are listed in the references for **NWDSList**, **NWDSRead**, and **NWDSSearch**.

NCP Calls

None

See Also

NWDSGetAttrName, **NWDSGetAttrVal**, **NWDSGetObjectName**, **NWDSList**, **NWDSSearch**

NWDSGetObjectHostServerAddress

Returns the addresses of the server where an object is located
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSGetObjectHostServerAddress (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              serverName,
   pBuf_T              netAddresses);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSGetObjectHostServerAddress
  (context : NWDSContextHandle;
   objectName : pnstr8;
   serverName : pnstr8;
   netAddresses : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of an NDS object.

serverName

(OUT) Points to the name of the server where an object is located.

netAddresses

(OUT) Points to a buffer containing the network addresses of the associated server.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetObjectHostServerAddress works only for objects having "Host Server" as an attribute (like Volume). Servers can have more than one address, such as an IPX and an IP address. *netAddresses* receives these addresses.

For more information, see Finding the Host Server of an Object.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSComputeAttrValSize, **NWDSGetAttrCount**, **NWDSGetAttrVal**

NWDSGetObjectName

Returns the name and information about the next object whose information is stored in a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetObjectName (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              objectName,
   pnuint32            attrCount,
   pObject_Info_T      objectInfo);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetObjectName
  (context : NWDSContextHandle;
   buf : pBuf_T;
   objectName : pnstr8;
   attrCount : pnuint32;
   objectInfo : pObject_Info_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

objectName

(OUT) Points to the name of the object whose information is at the current position in the buffer.

attrCount

(OUT) Points to the number of attributes following the object name.

objectInfo

(OUT) Points to additional information about the object (size of Object_Info_T).

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetObjectName should be called once for each object in the buffer. The count of objects whose information is stored in the buffer is determined by calling **NWDSGetObjectCount**.

NOTE: You must retrieve all of the information about the current object before calling **NWDSGetObjectName** for the next object.

buf points to a Buf_T filled in by **NWDSList**, **NWDSRead**, or **NWDSSearch**.

objectName points to the name of the current object in the buffer. The object's name is abbreviated if the context flag associated with DCV_CANONICALIZE_NAMES is set. Types are removed from the name if the flag associated with DCV_TYPELESS_NAMES is set.

You must allocate space for the object's name. The size of the allocated memory is ((MAX_DN_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

attrCount points to the number of attributes that follow the object name. The attribute count is always zero for a buffer returned by **NWDSList** since **NWDSList** only returns the names of objects. The attribute count will be zero or greater for a buffer returned by **NWDSSearch**.

objectInfo points to additional information about the object. You must allocate memory to retrieve this information (sizeof(Object_Info_T)).

The complete steps for removing information from a result buffer are shown in the reference listing for **NWDSList**, **NWDSRead**, and **NWDSSearch**.

NCP Calls

None

See Also

NWDSGetAttrName, **NWDSGetAttrVal**, **NWDSGetObjectCount**, **NWDSList**, **NWDSSearch**

NWDSGetObjectNameAndInfo

Returns the name and information about the next object whose information is stored in a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_GLOBAL_LIBRARY (NWDSSCODE) NWDSGetObjectNameAndInfo (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              objectName,
   pnuint32            attrCount,
   ppnstr8             objectInfo);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetObjectNameAndInfo (
   context : NWDSContextHandle;
   buf : pBuf_T;
   objectName : pnstr8;
   attrCount : pnuint32;
   objectInfo : ppnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer of a previous operation.

objectName

(OUT) Points to the name of the object whose information is at the current position in the buffer.

attrCount

(OUT) Points to the number of attributes following the object name.

objectInfo

(OUT) Points to additional information about the object.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetObjectNameAndInfo should be called once for each object in the buffer. The count of objects whose information is stored in the buffer is determined by calling the **NWDSGetObjectCount** function.

NOTE: You must retrieve all of the information about the current object before calling **NWDSGetObjectNameAndInfo** for the next object.

See DSI Flags.

The *buf* parameter points to a Buf_T filled in by the **NWDSList**, **NWDSRead**, **NWDSReadObjectDSIInfo**, or **NWDSSearch** functions.

The object name in the *objectName* parameter is abbreviated if the context flag associated with DCV_CANONICALIZE_NAMES is set. Types are removed from the name if the flag associated with DCV_TYPELESS_NAMES is set.

You must allocate space for the object name. The size of the allocated memory is ((MAX_DN_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

The attribute count in the *attrCount* parameter is always zero for a buffer returned by the **NWDSList** function since the **NWDSList** function only returns object information and not attribute information. The attribute count will be zero or greater for a buffer returned by the **NWDSSearch** or **NWDSRead** functions.

You must allocate memory to retrieve the information in the *objectInfo* parameter.

The complete steps for removing information from a result buffer are shown in the reference listing for the **NWDSList**, **NWDSRead**, and **NWDSSearch** functions.

NCP Calls

None

See Also

NWDSGetAttrName, **NWDSGetAttrVal**, **NWDSGetDSIInfo**, **NWDSGetObjectCount**, **NWDSReadObjectDSIInfo**, **NWDSList**, **NWDSRead**, **NWDSSearch**

NWDSGetPartitionExtInfo

Retrieves replica information from a result buffer filled by **NWDSListPartitions**

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSGetPartitionExtInfo (
   NWDSContextHandle   context,
   pnstr8              infoPtr,
   pnstr8              limit,
   nflag32             infoFlag,
   pnuint32            length,
   nptr                data);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetPartitionExtInfo (
   context : NWDSContextHandle;
   infoPtr : pnstr8;
   limit : pnstr8;
   infoFlag : nflag32;
   length : pnuint32;
   data : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

infoPtr

(IN) Points to the information returned from calling the **NWDSListPartitions** function.

limit

(IN) Points to the end of the buffer pointed to by the *infoPtr* parameter and used for overflow checking.

infoFlag

(IN) Specifies the data element to retrieve from the the buffer pointed to by the *infoPtr* parameter.

length

(IN) Points to the size of the returned information.

data

(IN) Points to the returned information.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Call NWDSGetPartitionExtInfo repeatedly to access all additional information requested in the DSP Flags passed into the *infoFlag* parameter.

For the complete steps for retrieving partition information see the **NWDSListPartitions** function.

NCP Calls

None

See Also

NWDSGetPartitionExtInfoPtr, **NWDSGetServerName**, **NWDSListPartitions**, **NWDSListPartitionsExtInfo**

NWDSGetPartitionExtInfoPtr

Retrieves a pointer to the replica information from a result buffer filled by **NWDSListPartitions**

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSGetPartitionExtInfoPtr (
   NWDSContextHandle   context,
   pBuf_T              buf,
   ppnstr8             infoPtr,
   ppnstr8             infoPtrEnd);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetPartitionExtInfoPtr (
   context : NWDSContextHandle;
   buf : pBuf_T;
   infoPtr : ppnstr8;
   infoPtrEnd : ppnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer received from calling the *NWDSListPartitions* function.

infoPtr

(OUT) Points to the returned information.

infoPtrEnd

(OUT) Points end of the returned information.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

See DSP Flags.

For the complete steps for retrieving partition information, see the **NWDSListPartitions** function.

NCP Calls

None

See Also

NWDSGetPartitionExtInfo, **NWDSGetServerName**, **NWDSListPartitions**, **NWDSListPartitionsExtInfo**

NWDSGetPartitionInfo

Retrieves replica information from a result buffer filled by **NWDSListPartitions**

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetPartitionInfo (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              partitionName,
   pnuint32            replicaType);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetPartitionInfo
  (context : NWDSContextHandle;
   buf : pBuf_T;
   partitionName : pnstr8;
   replicaType : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer to be read.

partitionName

(OUT) Points to the name of the root object of a partition.

replicaType

(OUT) Points to the replica type.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

buf points to a Buf_T filled by **NWDSListPartitions**.

partitionName points to a memory location containing the distinguished name of a partition for which replica information has been found. You must allocate space for the partition name. The size of the allocated memory is ((MAX_DN_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

replicatype points to NWREPLICA_TYPE containing information about the type of replica the partition is. The replica types are enumerated in NWDSDEFS.H as follows:

RT_MASTER	Master Replica
RT_SECONDARY	Secondary Replica
RT_READ_ONLY	Read-only Replica
RT_SUBREF	Partition with only a root object

For the complete steps for retrieving partition information see **NWDSListPartitions**.

NCP Calls

None

See Also

NWDSGetServerName, **NWDSListPartitions**

NWDSGetPartitionRoot

Returns the partition root name of the given object

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSGetPartitionRoot (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              partitionRoot);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSGetPartitionRoot
  (context : NWDSContextHandle;
   objectName : pnstr8;
   partitionRoot : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the object's name.

partitionRoot

(OUT) Points to the partition root name. You must allocate memory for *partitionRoot*; either MAX_DN_BYTES or MAX_DN_CHARS.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL

0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

If the object is itself a partition root, *partitionRoot* is the same as the object name.

The caller must allocate space for *partitionRoot*. The size of the memory allocated is ((MAX_DN_CHARS)+1)*sizeof(character size), where character size is 1 for single-byte characters and 2 for double-byte characters (Unicode is double byte). One character is used for NULL termination.

NWDSGetServerAddresses

Returns the network addresses of the server associated with a connection handle

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSGetServerAddresses (
   NWDSContextHandle   context,
   NWCONN_HANDLE       conn,
   pnuint32            countNetAddress,
   pBuf_T              netAddresses);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSGetServerAddresses
  (context : NWDSContextHandle;
   conn : NWCONN_HANDLE;
   countNetAddress : pnuint32;
   netAddresses : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS Access context for the request.

conn

(IN) Specifies the connection handle for the target server.

countNetAddress

(OUT) Points to the number of network addresses contained in *netAddresses*.

netAddresses

(OUT) Points to a buffer containing the network address associated with the server.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative values	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Servers can have more than one address, such as an IPX and an IP address. *netAddresses* receives these addresses.

For more information, see Retrieving Addresses of a Connected Server.

NCP Calls

0x2222 104 02   Send NDS Fragmented Request/Reply
53   Get Server Address

See Also

NWDSComputeAttrValSize, **NWDSGetAttrCount**, **NWDSGetAttrVal**

NWDSGetServerDN

Returns the server's distinguished name in NDS
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSGetServerDN (
   NWDSContextHandle   context,
   NWCONN_HANDLE       conn,
   pnstr8              serverDN);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSGetServerDN
  (context : NWDSContextHandle;
   conn : NWCONN_HANDLE;
   serverDN : pnstr8
) : NWDSCCODE;
```

Parameters

context
(IN) Specifies the NDS context for the request.

conn
(IN) Specifies the connection to the server to be queried.

serverDN
(OUT) Points to the distinguished name of the server.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

value	-699).

Remarks

conn is the connection handle to the server.

The caller must allocate space to hold the distinguished name of the server and set *serverDN* to point to it. The size of the allocated memory is (MAX_DN_CHARS+1)*sizeof(character size) where character size is 1 for single-byte characters and 2 for double-byte characters (Unicode is double byte). One character is used for NULL termination.

Whether the server name is returned as a complete name or a partial name depends upon the setting of the context flag associated with DCV_CANONICALIZE_NAMES.

NWDSGetServerDN does not work on a local server with a connection 0. Call **AttachToFileServer** then **GetCurrentConnection** and pass the returned value to **NWDSGetServerDN** to return the server's DN. If connection 0 is used, a -333 error is returned.

NCP Calls

0x2222 104 02 Send NDS Fragmented Request/Reply
53 Get Server Address

NWDSGetServerName

Returns the name of the current server, as well as the number of partitions on the server, from a result buffer

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetServerName (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              serverName,
   pnuint32            partitionCount);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetServerName
  (context : NWDSContextHandle;
   buf : pBuf_T;
   serverName : pnstr8;
   partitionCount : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

serverName

(OUT) Points to the server name.

partitionCount

(OUT) Points to the number of partition names in the result buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSGetServerName should be called first to read from a result buffer returned by **NWDSListPartitions**.

buf points to a Buf_T filled by **NWDSListPartitions**.

serverName points to a memory location containing the distinguished name of the server for which replica information has been found. You must allocate space for the server name. The size of the allocated memory is ((MAX_DN_CHARS)+1*)sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

For the complete steps for retrieving partition information see **NWDSListPartitions**.

NCP Calls

None

See Also

NWDSGetPartitionInfo, **NWDSListPartitions**

NWDSGetSyntaxCount

Returns the number of NDS syntaxes whose information is stored in a result buffer filled by **NWDSReadSyntaxes**

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetSyntaxCount (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            syntaxCount);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetSyntaxCount
  (context : NWDSContextHandle;
   buf : pBuf_T;
   syntaxCount : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the buffer being read.

syntaxCount

(OUT) Points to the number of syntaxes stored in the buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Before reading the syntax information from a result buffer filled by **NWDSReadSyntaxes**, you must first call **NWDSGetSyntaxCount** to determine the number of syntaxes whose information is stored in the buffer.

When **NWDSGetSyntaxCount** returns, the location pointed to by *syntaxCount* specifies the number of syntaxes whose information is stored in the buffer. To remove the syntax information from the result buffer, call **NWDSGetSyntaxDef** once for each syntax whose information is stored in the buffer.

The complete steps for retrieving information about the syntaxes in the NDS Schema are listed in the reference for **NWDSReadSyntaxes**.

NCP Calls

None

See Also

NWDSGetSyntaxDef, **NWDSReadSyntaxes**

NWDSGetSyntaxDef

Retrieves the next NDS-syntax definition from a result buffer filled by **NWDSReadSyntaxes**

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSGetSyntaxDef (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              syntaxName,
   pSyntax_Info_T      syntaxDef);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSGetSyntaxDef
  (context : NWDSContextHandle;
   buf : pBuf_T;
   syntaxName : pnstr8;
   syntaxDef : pSyntax_Info_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

syntaxName

(OUT) Points to the name of the syntax whose definition is stored at the current position in the result buffer.

syntaxDef

(OUT) Points to the syntax definition.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Before the initial call to **NWDSGetSyntaxDef**, call **NWDSGetSyntaxCount** to determine the number of syntaxes whose information is stored in the result buffer. Then call **NWDSGetSyntaxDef** once for each syntax whose information is stored in the result buffer.

buf points to a result buffer containing information about syntaxes. This result buffer is allocated by **NWDSAllocBuf** and filled by **NWDSReadSyntaxes**.

syntaxName points to the name of the attribute whose definition is in the result buffer. The user must allocate memory to store the name. The size of the allocated memory is ((MAX_SCHEMA_NAMES_CHARS)+1)*sizeof(character size), where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

syntaxDef points to the remainder of the syntax definition. If **NWDSReadSyntaxes** was called with a request for syntax names only (DS_SYNTAX_NAMES), *syntaxDef* is ignored by **NWDSGetSyntaxDef** and can be NULL.

The user must allocate memory, sizeof(Syntax_Info_T), to receive the syntax definition.

The complete steps for retrieving information about the syntaxes in the NDS Schema are listed in the reference for **NWDSReadSyntaxes**.

NCP Calls

None

See Also

NWDSGetSyntaxCount, **NWDSReadSyntaxes**

NWDSGetSyntaxID

Returns the syntax ID of a given attribute

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSGetSyntaxID (
   NWDSContextHandle   context,
   pnstr8              attrName,
   pnuint32            syntaxID);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSGetSyntaxID
  (context : NWDSContextHandle;
   attrName : pnstr8;
   syntaxID : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

attrName

(IN) Points to the attribute name whose syntax ID you want to determine.

syntaxID

(OUT) Points to the syntax ID of the attribute.

Return Values

0x0000	SUCCESSFUL
negative	Negative values indicate errors. See NDS Values (-001 to

negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Syntax IDs are enumerated in nwdsdefs.h. A description of syntax definitions can be found in NDS Attribute Syntax Definitions.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

NWDSInitBuf

Initializes a buffer for use as a request buffer for an NDS function
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSInitBuf (
   NWDSContextHandle   context,
   nuint32             operation,
   pBuf_T              buf);
```

Pascal Syntax

```
#include <nwdsbuft.h>

Function NWDSInitBuf
  (context : NWDSContextHandle;
   operation : nuint32;
   buf : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

operation

(IN) Specifies the NDS operation for which the buffer is being initialized.

buf

(IN) Points to the buffer being initialized.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Only request buffers need to be initialized. Result buffers do not require initialization.

First allocate the request buffer by calling **NWDSAllocBuf**. Then call **NWDSInitBuf** to initialize the buffer for a particular type of operation.

operation indicates the operation for which the buffer will be used:

	Name	Related Function(s)
3	DSV_READ	**NWDSExtSyncRead** **NWDSListAttrsEffectiveRights** **NWDSRead** **NWDSReadReferences**
4	DSV_COMPARE	**NWDSCompare**
6	DSV_SEARCH	**NWDSExtSyncList** **NWDSExtSyncSearch** **NWDSListByClassAndName** **NWDSListContainers** **NWDSPutFilter** **NWDSSearch**
7	DSV_ADD_ENTRY	**NWDSAddObject**
9	DSV_MODIFY_ENTRY	**NWDSModifyObject**
12	DSV_READ_ATTR_DEF	**NWDSReadAttrDef**
14	DSV_DEFINE_CLASS	**NWDSDefineClass**
15	DSV_READ_CLASS_DEF	**NWDSReadClassDef**
16	DSV_MODIFY_CLASS_DEF	**NWDSModifyClassDef**
18	DSV_LIST_CONTAINABLE_CLASSES	**NWDSListContainableClasses**
40	DSV_READ_SYNTAXES	**NWDSGetSyntaxDef**, **NWDSPutSyntaxName**, and **NWDSReadSyntaxes**

buf is updated to reflect the selected operation.

NCP Calls

None

See Also

NWDSAllocBuf, **NWDSFreeBuf**

NWDSInspectEntry

Inspects an object for correctness

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSInspectEntry (
   NWDSContextHandle   context,
   pnstr8              serverName,
   pnstr8              objectName,
   pBuf_T              errBuffer);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSInspectEntry
  (context : NWDSContextHandle;
   serverName : pnstr8;
   objectName : pnstr8;
   errBuffer : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

serverName

(IN) Points to the server name to which to connect.

objectName

(IN) Points to the object name to be inspected.

errBuffer

(OUT) Points to the Buf_T structure which is a result buffer containing the requested information.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSInspectEntry is a diagnostic function allowing you to inspect an object in the replica(s) on a specific server to see if the object needs to be repaired.

If no partition exists on the server specified by *serverName* or the object does not exist in the partition(s) on the specified server, **NWDSInspectEntry** returns ERR_NO_SUCH_ENTRY.

Contents of the Output Buffer

After successful completion of this function call, the output buffer contains the following data:

Return Code	nuint	Success
Entry Size	nuint	Total number of bytes occupied by the entry's base record and its attribute values
Error Count	nuint	Number of errors found in the entry record on that server
Error Reports	nuint	List of error codes indicating errors in the entry record

NCP Calls

0x2222 23 17 Get File Server Information

0x2222 23 22 Get Station's Logged Info (old)

0x2222 23 28 Get Station's Logged Info

0x2222 104 01 Ping for NDS NCP

0x2222 104 02 Send NDS Fragmented Request/Reply

NWDSJoinPartitions

Joins a subordinate partition to its parent partition
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSJoinPartitions (
   NWDSContextHandle   context,
   pnstr8              subordinatePartition,
   nflag32             flags);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSJoinPartitions
  (context : NWDSContextHandle;
   subordinatePartition : pnstr8;
   flags : nflag32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

subordinatePartition

(IN) Points to the name of the subordinate partition to be joined.

flags

Reserved; pass in NULL.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

For partitions to be joined, a single replica (Master) of both the parent and subordinate partitions must exist. In addition, the master replicas must exist on the same server.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAddReplica, **NWDSChangeReplicaType**, **NWDSSplitPartition**

NWDSList

Lists the immediate subordinates of an object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSList (
   NWDSContextHandle   context,
   pnstr8              object,
   pnint32             iterationHandle,
   pBuf_T              subordinates);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSList
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint32;
   subordinates : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

object

(IN) Points to the name of the object whose immediate subordinates are to be listed.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSList**.

subordinates

(OUT) Points to a result buffer containing an Object_Info_T structure for each subordinate.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSList succeeds if the object specified by *object* is found in NDS, regardless of whether there is any subordinate information to return.

See DSI Flags.

If the name pointed to by the *object* parameter involves one or more aliases, the aliases are dereferenced unless prohibited by the context flag associated with DCV_DEREF_ALIAS.

The results buffer pointed to by *subordinates* receives a sequence of Object_Info_T structures containing information about objects subordinate to the specified object.

iterationHandle controls the retrieval of list results larger than the result buffer pointed to by *subordinates*.

Before the initial call to **NWDSList**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

When **NWDSList** returns from its initial call, if the result buffer holds the complete results, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSList** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

To end the List operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_LIST to free memory and states associated with the List operation.

For more information, see Listing Objects in an NDS Container.

> **NOTE:** On large networks, iterative processes, like **NWDSList**, might take a lot of time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. These processes can be interrupted or aborted using **NWDSCloseIteration**.
>
> Developers should use **NWDSCloseIteration** to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSSearch**

NWDSListAttrsEffectiveRights

Returns an object's effective privileges on another object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsacl.h>

NWDSCCODE N_API NWDSListAttrsEffectiveRights (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              subjectName,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint32             iterationHandle,
   pBuf_T              privilegeInfo);
```

Pascal Syntax

```
#include <nwdsacl.inc>

Function NWDSListAttrsEffectiveRights
  (context : NWDSContextHandle;
   objectName : pnstr8;
   subjectName : pnstr8;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint32;
   privilegeInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the NDS object whose access rights are to be checked.

subjectName

(IN) Points to the name of the NDS object to which the privileges are assigned.

allAttrs

(IN) Specifies whether all attributes should be returned.

attrNames

(IN) Points to a request buffer containing the names of the attribute definitions for which information is to be returned.

iterationHandle

(IN/OUT) Points to the information needed to resume subsequent iterations of **NWDSListAttrsEffectiveRights**.

privilegeInfo

(OUT) Points to a result buffer receiving the requested attribute names and privileges.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

subjectName is the name of a directory object. If *subjectName* is NULL, the name of the currently logged-in object is used.

allAttrs and *attrNames* indicate which attributes you are requesting privileged information about. If *allAttrs* is TRUE, privileged information about all optional and mandatory attributes defined for the base class of the object are returned. NULL can also be passed for *attrNames* when *allAttrs* is TRUE. If *allAttrs* is FALSE, privileged information is returned only about the attributes named in the buffer pointed to by *attrNames*.

attrNames points to a request buffer explicitly specifying the names of the attributes for which information is to be returned.

iterationHandle controls retrieval of list results larger than the result buffer pointed to by *attrNames*.

Before the initial call to **NWDSListAttrsEffectiveRights**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when **NWDSListAttrsEffectiveRights** returns from its initial call, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSListAttrsEffectiveRights** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

For more information, see Determining the Effective Rights of an Object.

NOTE: On large networks, iterative processes, like **NWDSListAttrsEffectiveRights**, might take a lot of time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. These processes can be interrupted or aborted using **NWDSCloseIteration**.

Developers should use **NWDSCloseIteration** to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSAllocBuf**, **NWDSGetAttrVal**, **NWDSInitBuf**

NWDSListByClassAndName

Lists the immediate subordinates for an NDS object and restricts the list to subordinate objects matching a specified object class and/or name

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSListByClassAndName (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              className,
   pnstr8              subordinateName,
   pnint32             iterationHandle,
   pBuf_T              subordinates;)
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSListByClassAndName
   (context : NWDSContextHandle;
    objectName : pnstr8;
    className : pnstr8;
    subordinateName : pnstr8;
    iterationHandle : pnint32;
    subordinates : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object whose subordinates are to be listed.

className

(IN) Points to a class name to be used as a filter. This can be NULL.

subordinateName

(IN) Points to an object name to be used as a filter. This can be NULL.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSListByClassAndName** (set to NO_MORE_ITERATIONS initially).

subordinates

(OUT) Points to a result buffer containing a list of subordinate objects matching the search criteria.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
0xFE0D	UNI_NO_DEFAULT
0xFE0F	UNI_HANDLE_MISMATCH
0xFEB5	ERR_NULL_POINTER
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSListByClassAndName controls the list output with filters on the class and/or name.

If the context flag associated with DCV_TYPELESS_NAMES is set, the returned list of object names in the buffer will be typeless. If the flag is off, the returned list will contain typed names.

The name given for *className*'s filter is the name of an object class, such as User, Computer, or Server.

The value given for *subordinateName*'s filter can be one of the following:

The left-most name of an object, such as Adam or Graphics Printer.
A string with asterisks (*), such as A* or Gr*.
NULL, which means any name is valid.

The location of the subordinate object(s) in the NDS tree is immediately subordinate to the object specified by *objectName*. It is not relative to the current name context in NDS specified by *context*.

The relationship between *className* and *subordinateName* is an "AND" relationship.

When *className* and *subordinateName* are provided, a list of immediate subordinate objects restricted by both filters is returned.

When *className* is NULL and *subordinateName* is NULL, a list of all immediate subordinates is returned.

When *className* is provided and *subordinateName* is NULL, a list of immediate subordinates restricted only by *className*'s filter is returned.

When *className* is NULL and *subordinateName* is provided, a list of immediate subordinates restricted only by *subordinateName*'s filter is returned.

The following examples show how to use wildcards for untyped names:

c*	Any object whose left-most name begins with a "c" character.
M*y	Any object beginning with "M" and ending with"y" such as Mary.

If the wildcard name specified for *subordinateName* includes a type, such as "CN," the name must include the equals (=) sign. The following examples show how to use wildcards for typed names:

cn=*	Any object whose left-most name is a common name.
cn=c*	Any object whose left-most name is a common name and begin with "c."
o*=*	Any object whose left-most name has a naming attribute beginning with an "o," such as O or OU.
o*=c*	Any object whose left-most name has a naming attribute beginning with an "o," and whose name begins with "c."

iterationHandle controls retrieval of search results larger than the result buffer pointed to by *subordinates*.

Before the initial call to **NWDSListByClassAndName**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when **NWDSListByClassAndName** returns from its initial call, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If the *iterationHandle* is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSListByClassAndName** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

To end the List operation before the complete results have been retrieved, call **NWDSListByClassAndName** with a value of DSV_SEARCH to free memory and states associated with the List operation.

Allocate the result buffer pointed to by *subordinates*, by calling **NWDSAllocBuf**. This result buffer does not need to be initialized because it is a result buffer. For more information, see Retrieving Results from NDS Output Buffers.

NOTE: On large networks, iterative processes, like **NWDSListByClassAndName**, might take a lot of time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. These processes can be interrupted or aborted using **NWDSCloseIteration**.

Developers should use **NWDSCloseIteration** to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSList**

NWDSListContainableClasses

Returns the names of the object classes that can be contained by (subordinate to) the specified object in the NDS tree

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdssch.h>

NWDSCCODE N_API NWDSListContainableClasses (
   NWDSContextHandle   context,
   pnstr8              parentObject,
   pnint32             iterationHandle,
   pBuf_T              containableClasses);
```

Pascal Syntax

```
#include <nwdssch.inc>

Function NWDSListContainableClasses
  (context : NWDSContextHandle;
   parentObject : pnstr8;
   iterationHandle : pnint32;
   containableClasses : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

parentObject

(IN) Points to the name of the parent object for which containable classes are to be listed.

iterationHandle

(IN/OUT) Points to the information needed to resume subsequent iterations of **NWDSListContainableClasses**.

containableClasses

(OUT) Points to a buffer containing the names of object classes contained by the specified parent object.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSListContainableClasses can be used to build a list of object classes that can be used to create objects subordinate to the parent object specified by *parentObject*.

parentObject points to the name of an NDS object for which containable object-classes are to be listed. If this parent object is not a valid container object, an error is returned.

iterationHandle controls retrieval of results larger than the buffer pointed to by *containableClasses*.

Before the initial call to **NWDSListContainableClasses**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

When **NWDSListContainableClasses** returns from its initial call, if the result buffer holds the complete results, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If *iterationHandle* is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent class to **NWDSListContainableClasses** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

For more information, see Listing Containable Classes.

> **NOTE:** To end the List operation before all of the results have been retrieved, call **NWDSCloseIteration** with a value of DSV_LIST_CONTAINABLE_CLASSES to free memory and states associated with **NWDSListContainableClasses**.

The level of granularity for partial results (those split across multiple iterations) is an individual class name.

containableClasses points to a result buffer that receives the list of names of object classes that can be used to create objects contained by the specified parent object. The result buffer contains the names of only the object classes marked as effective in the NDS Schema (those from which objects can be created). Alias is always included in the list.

> **NOTE:** On large networks, iterative processes, like **NWDSListContainableClasses**, might take a lot of time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. These processes can be interrupted or aborted using **NWDSCloseIteration**.
>
> Developers should use **NWDSCloseIteration** to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSAddObject**

NWDSListContainers

Lists container objects subordinate to a specific NDS object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSListContainers (
   NWDSContextHandle   context,
   pnstr8              object,
   pnint32             iterationHandle,
   pBuf_T              subordinates);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSListContainers
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint32;
   subordinates : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

object

(IN) Points to the name of the object whose subordinate container objects are to be listed.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSListContainers**. This should be initially set to NO_MORE_ITERATIONS.

subordinates

(OUT) Points to a result buffer containing a list of subordinate container objects.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
0xFE01	ERR_BAD_CONTEXT
0xFE0D	UNI_NO_DEFAULT
0xFE0F	UNI_HANDLE_MISMATCH
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The name specified by *objectName* is relative to the current name context in *context*. It can be typed or untyped.

iterationHandle controls retrieval of search results larger than the result buffer pointed to by *subordinates*.

Before the initial call to **NWDSListContainers**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when **NWDSListContainers** returns from its initial call, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If *iterationHandle* is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSListContainers** to obtain further portions of the results. When the results are completely retrieved, the contents of *iterationHandle* will be set to NO_MORE_ITERATIONS.

To end the List operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_SEARCH to free memory and states associated with the List operation.

The contents of the result buffer pointed to by *subordinates* are overwritten with each subsequent call to **NWDSListContainers**. Remove the contents from the result buffer before each subsequent call to **NWDSListContainers**.

Allocate the result buffer pointed to by *subordinates*, by calling **NWDSAllocBuf**. This result buffer does not need to be initialized because it is a result buffer. For more information, see Retrieving Results from NDS Output Buffers.

NOTE: On large networks, iterative processes, like **NWDSListContainers**, might take a lot of time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. These processes can be interrupted or aborted using **NWDSCloseIteration**.

Developers should use **NWDSCloseIteration** to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSList**

NWDSListPartitions

Returns information about the replicas of partitions stored on the specified server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSListPartitions (
   NWDSContextHandle   context,
   pnint32             iterationHandle,
   pnstr8              server,
   pBuf_T              partitions);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSListPartitions (
   context : NWDSContextHandle;
   iterationHandle : pnint32;
   server : pnstr8;
   partitions : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of the operation.

server

(IN) Points to the server name whose list of partitions is requested.

partitions

(OUT) Points to a result buffer that receives the name and replica type for each partition stored on the specified server.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

For more information, see Retrieving Partition Information from a Server.

NOTE: Iterative processes, like **NWDSListPartitionsExtInfo**, can be interrupted or aborted by calling the **NWDSCloseIteration** function.

Developers should call the **NWDSCloseIteration** function to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSGetServerName**, **NWDSGetPartitionInfo**

NWDSListPartitionsExtInfo

Returns information about the replicas of partitions stored on the specified server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_GLOBAL_LIBRARY (NWDSCCODE) NWDSListPartitionsExtInfo (
   NWDSContextHandle   context,
   pnint32             iterationHandle,
   pnstr8              server,
   nflag32             DSPFlags,
   pBuf_T              partitions);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSListPartitionsExtInfo
  (context : NWDSContextHandle;
   iterationHandle : pnint32;
   server : pnstr8;
   DSPFlags : nflag32
   partitions : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of the operation.

server

(IN) Points to the server name whose list of partitions is requested.

DSPFlags

(IN) Points to the DSP flags.

partitions

(OUT) Points to a result buffer that receives the name and replica type for each partition stored on the specified server.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

For more information, see Retrieving Partition Information from a Server and DSP Flags.

NOTE: Iterative processes, like **NWDSListPartitionsExtInfo**, can be interrupted or aborted by calling the **NWDSCloseIteration** function.

Developers should call the **NWDSCloseIteration** function to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSGetPartitionExtInfo**, **NWDSGetPartitionExtInfoPtr**, **NWDSGetServerName**, **NWDSListPartitions**

NWDSLockConnection (obsolete 6/96)

Removes the connection from the Least Recently Used (LRU) list and licenses the connection if it is authenticated but is now obsolete. Call **NWCCLicenseConn** instead.

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWDSLockConnection
   (NWCONN_HANDLE   conn);
```

Pascal Syntax

```
Function NWDSLockConnection
   (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the connection handle to lock.

Return Values

These are common return values; see Return Values for more information.

0x000 0	SUCCESSFUL

Remarks

NWDSLockConnection (obsolete 6/96) is not supported on some requesters. However, there will still be control over the connection slot as long as the connection handle is not closed.

NWDSLockConnection (obsolete 6/96) can be called several times without changing the state of the connection even if the connection is already licensed.

If the connection is autheticated to a 4.0 NetWare server and is not licensed, the connection will be licensed.

If **NWDSUnlockConnection** is called and another task does not have any resouces on this connection, the connection will be unlicensed on the NetWare server.

NCP Calls

None

NWDSLogin

Performs all authentication operations needed to establish a client's connection to the network and to the network's authentication service

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsasa.h>

NWDSCCODE N_API NWDSLogin (
   NWDSContextHandle   context,
   nflag32             optionsFlag,
   pnstr8              objectName,
   pnstr8              password,
   nuint32             validityPeriod);
```

Pascal Syntax

```
#include <nwdsasa.inc>

Function NWDSLogin
  (context : NWDSContextHandle;
   optionsFlag : nflag32;
   objectName : pnstr8;
   password : pnstr8;
   validityPeriod : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

optionsFlag

Reserved; pass in zero.

objectName

(IN) Points to the name of the object logging into the network.

password

(IN) Points to the client's password.

validityPeriod

Reserved for future use to indicate, in seconds, the period during

Reserved for future use to indicate, in seconds, the period during which authentication will be valid with other servers. Pass in zero (0).

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSLogin caches authentication information locally to be used by other functions and in background authentication to additional services.

password points to the client's current password in clear text. If there is no password for the client, it's value should point to a zero-length string ("").

validityPeriod specifies the time interval during which the client's authentication information remains valid. If the value is 0, the authentication service supplies a default value. Also, if this value exceeds the default value supplied by the authentication service, the default value is applied.

The validation period begins by calling **NWDSLogin**. The minimum recommended period is 60 seconds. Shorter times may cause the authenticator to expire before it can be used. If the authenticator expires before the client logs out, the log out process is NOT completed.

If an application has a local copy of any password value, the value should be erased as soon as possible to prevent compromising the security of the password.

Until an authenticated connection is established, the client can access only NDS information classified as public.

NCP Calls

None

See Also

NWDSAuthenticate, **NWDSLogout**

NWDSLogout

Terminates a client's connection to the network and invalidates any information cached locally by **NWDSLogin**

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsasa.h>

NWDSCCODE N_API NWDSLogout (
   NWDSContextHandle   context);
```

Pascal Syntax

```
#include <nwdsasa.inc>

Function NWDSLogout
  (context : NWDSContextHandle
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

After calling **NWDSLogout**, new connections cannot be established by calling **NWDSAuthenticate**. **NWDSLogout** leaves intact all server attachments and other session connections, authenticated or unauthenticated.

NWDSLogout invalidates the cached authenticator even if the function results in an error.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAuthenticate, **NWDSLogin**

NWDSMapIDToName

Returns the directory name for an object denoted by a connection handle and an object ID

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSMapIDToName (
   NWDSContextHandle   context,
   NWCONN_HANDLE       conn,
   nuint32             objectID,
   pnstr8              object);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSMapIDToName
  (context : NWDSContextHandle;
   conn : NWCONN_HANDLE;
   objectID : nuint32;
   objectName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

conn

(IN) Specifies the connection handle for the target server.

objectID

(IN) Specifies the object ID.

object

(OUT) Points to the object's distinguished name.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

conn contains a server connection handle. This identifies the server from which the object ID was obtained.

objectID contains the object ID returned by the specified server.

object receives the name of the NDS object corresponding to the given object ID. The caller must allocate memory to hold the object's name. The size of the memory allocated is (MAX_DN_CHARS+1)*sizeof(character size), where character size is 1 for single-byte characters and 2 for double-byte characters (Unicode is double byte). One character is used for NULL termination.

Since object IDs are unique only in relation to a particular server, the use of object IDs is restricted to the server from which they originate. An object ID returned by one server is meaningless to another server. Furthermore, a returned object ID may be valid only for a short period of time.

For these reasons, applications should not store object IDs locally. Rather, they should store the full name of an NDS object. (If an application needs a short-hand representation of an object, it should manage its own local name-to-ID mapping.)

NCP Calls

0x2222 104 02   Send NDS Fragmented Request/Reply

See Also

NWDSMapNameToID

NWDSMapNameToID

Returns the object ID for an NDS object on a specified server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSMapNameToID (
   NWDSContextHandle   context,
   NWCONN_HANDLE       conn,
   pnstr8              object,
   pnuint32            objectID);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSMapNameToID
  (context : NWDSContextHandle;
   conn : NWCONN_HANDLE;
   objectName : pnstr8;
   objectID : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

conn

(IN) Specifies the connection handle for target server.

object

(IN) Points to the NDS object name.

objectID

(OUT) Points to the object ID for the specified object.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The returned ID can be used as input to older routines which require an *objectID* (for example, **NWAddTrustee**).

Since object IDs are unique only in relation to a particular server, the use of object IDs is restricted to the server from which they originate. An object ID returned by one server is meaningless to another server. Furthermore, a returned object ID may be valid only for a short period of time.

For these reasons, applications should not store object IDs locally. Rather, they should store the full name of an NDS object. (If an application needs a short-hand representation of an object, it should manage its own local name-to-ID mapping.)

conn contains a server connection handle. It identifies the server from which the object ID is to be obtained.

object points to the name of the NDS object for which the ID is to be returned.

It is not necessary for the object to be defined in a partition replica stored on the target server. If the object is not stored on the server, the server generates a temporary reference to the object and returns the ID for reference to the client.

objectID points to the object ID of the specified name on the server identified by *conn*.

NCP Calls

0x2222 104 02   Send NDS Fragmented Request/Reply

See Also

NWDSMapIDToName

NWDSModifyClassDef

Modifies an existing object-class definition
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdssch.h>

NWDSCCODE N_API NWDSModifyClassDef (
   NWDSContextHandle   context,
   pnstr8              className,
   pBuf_T              optionalAttrs);
```

Pascal Syntax

```
#include <nwdssch.inc>

Function NWDSModifyClassDef
  (context : NWDSContextHandle;
   className : pnstr8;
   optionalAttrs : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

className

(IN) Points to the object class name whose definition is to be modified.

optionalAttrs

(IN) Points to a request buffer containing the names of attributes to be added to the object-class definition's Optional Attribute Names list.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The only modifications clients can make to existing object-class definitions is the addition of optional attributes. No other characteristic of the object-class definition can be changed.

className identifies the object class to which optional attributes will be added.

optionalAttrs points to a request buffer containing a list of attribute names to be added to the Optional Attribute Names list of the object-class' definition.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSDefineClass

NWDSModifyDN

Changes the distinguished name of an object or its alias in the NDS tree
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSModifyDN (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              newDN,
   nbool8              deleteOldRDN);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSModifyDN
  (context : NWDSContextHandle;
   objectName : pnstr8;
   newDN : pnstr8;
   deleteOldRDN : nbool8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the object's old name.

newDN

(IN) Points to the object's new name.

deleteOldRDN

(IN) Specifies whether to discard the old DN. If FALSE, the old DN is retained as an additional attribute value.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The DN is the name of the object relative to its superior in the NDS tree. The object being modified must be a leaf object, but it may be either an object or its alias.

objectName points to the object whose DN is to be modified. Aliases in the name will not be dereferenced.

newDN specifies the new DN of the object. It is a string identifying the name's attribute type and the attribute value in the form:

```
attribute type = attribute value
```

For example:

```
"CN = Mary"
```

If an attribute value in the new DN does not already exist in the object, it is added. If it cannot be added, an error is returned.

If *deleteOldDN* is TRUE, all attribute values in the old DN that are not in the new DN are deleted. If FALSE, old values remain in the object (but not as a part of the DN). The flag must be TRUE where a single-value attribute in the DN has its value changed by **NWDSModifyDN**.

If **NWDSModifyDN** removes the last attribute value of an attribute while identifying a new attribute for the DN, the old attribute is deleted.

Aliases are never dereferenced by **NWDSModifyDN**. The context flag associated with DCV_DEREF_ALIASES is not relevant to **NWDSModifyDN** and is ignored.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSModifyObject, **NWDSSetContext**, **NWDSGetContext**

NWDSModifyObject

Modifies an object or its alias
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSModifyObject (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnint32             iterationHandle,
   nbool8              more,
   pBuf_T              changes);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSModifyObject
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint32;
   more : nbool8;
   changes : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the object name to be modified.

iterationHandle

(IN) Points to the iteration number (-1 initially).

more

(IN) Specifies whether additional information will be returned:

0	No more information

0	No more information
nonzero	More information will be returned

changes

(IN) Points to the set of changes to be applied to the object.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSModifyObject cannot modify an object's RDN. It can perform only the following:

Add a new attribute

Remove an attribute

Add values to an attribute

Remove values from an attribute

Replace the values of an attribute

objectName identifies the object to be modified. The object can be an alias. Any aliases in the name are not dereferenced.

The *changes* parameter defines a sequence of modifications, which are applied in the order specified. The buffer is allocated by calling **NWDSAllocBuf** and initialized for DSV_MODIFY_ENTRY by calling **NWDSInitBuf**. The specified changes are inserted into the buffer by calling **NWDSPutChange** and **NWDSPutAttrVal**.

> **NOTE:** If the *iterationHandle* parameter is set to 0 initially, **NWDSModifyObject** will ignore the value and process the request as if -1 was passed.

If the *more* parameter is set to nonzero, **NWDSModifyObject** will perform the necessary steps to iteratively call itself.

In order to iteratively call **NWDSModifyObject**, the DS.NLM file must support the iteration feature or ERR_BUFFER_FULL will be returned.

If any of the individual modifications fail, an error is generated and the object is left in the state it was prior to the operation. Furthermore, the end result of the sequence of modifications may not violate the NDS schema. (However, it is possible, and sometimes necessary, for the individual object modification changes to appear to do so.) If an attempt is made to modify the object class attribute, an error is returned.

Change records are inserted into the buffer by calling **NWDSPutChange**. Each type of change modification is explained below:

0	#00	DS_ADD_ATTRIBUTE: New attribute to be added to the object (attempting to add an already-existing attribute results in an error)
1	#01	DS_REMOVE_ATTRIBUTE: Attribute to be removed from the object (attempting to remove a non-existing attribute results in an error and not allowed if the attribute is present in the RDN)
2	#02	DS_ADD_VALUE: Values to be added to an attribute (attempting to add an already-existing value or add a value to a nonexistent attribute results in an error)
3	#03	DS_REMOVE_VALUE: Values to be removed from an attribute (attempting to delete nonexistent values results in an error and not allowed if any of the values is present in the RDN)

Values may be replaced by a combination of Remove Values and Add Values operations in a single call to **NWDSModifyObject**. Aliases are never dereferenced by **NWDSModifyObject**. The setting of the context flag associated with DCV_DEREF_ALIASES is not relevant to **NWDSModifyObject** and is ignored.

NCP Calls

0x2222 23 17 Get File Server Information

0x2222 23 22 Get Station's Logged Info (old)

0x2222 23 28 Get Station's Logged Info

0x2222 104 01 Ping for NDS NCP

0x2222 104 02   Send NDS Fragmented Request/Reply

See Also

NWDSModifyDN, **NWDSRemoveObject**, **NWDSModifyRDN**

NWDSModifyRDN

Changes the least significant name of an NDS object or its alias in the NDS tree

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSModifyRDN (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              newDN,
   nbool8              deleteOldRDN);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSModifyRDN
  (context : NWDSContextHandle;
   objectName : pnstr8;
   newDN : pnstr8;
   deleteOldRDN : nbool8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the object's current name.

newDN

(IN) Points to the object's new name.

deleteOldRDN

(IN) Specifies whether to discard the old RDN. If FALSE, the old RDN is retained as an additional attribute value.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSModifyRDN does not move an object to a new location in the NDS tree.

NWDSModifyRDN only changes the least significant (left-most) name in a leaf object's distinguished name. It does not change an object's more significant names, since changing those names changes the location of the object in the NDS tree. For example, if the object's name is

```
CN=Hector.OU=Graphics.O=WimpleMakers
```

you can change the common name "CN=Hector" to "CN=Duke" since it is a leaf object. However, you cannot change the "OU=Graphics" to "OU=Marketing" since that would change the location of the object (Hector) within the NDS tree. (To move an object call **NWDSMoveObject**.)

You cannot change the name of an object that is not a leaf node. For example, in the above case you cannot change the name of

```
OU=Graphics.O=WimpleMakers
```

to

```
OU=Presentation Graphics.O=WimpleMakers
```

because Graphics is not a leaf node; it contains the subordinate object because Graphics is not a leaf node; it contains the subordinate object named Hector.

objectName identifies the object whose name is to be modified. Aliases in the name are not dereferenced.

newDN specifies the new name of the object. If an attribute value in the new DN does not already exist in the object, it is added. If it cannot be added, an error is returned.

If *deleteOldRDN* is TRUE, all attribute values in the old DN that are not in the new DN are deleted. If FALSE, old values remain in the object (but not as a part of the DN). *deleteOldRDN* must be TRUE where a single-value attribute in the DN has its value changed by **NWDSModifyRDN**.

If **NWDSModifyRDN** removes the last attribute value of an attribute while identifying a new attribute for the DN, the old attribute is deleted.

Aliases are never dereferenced by **NWDSModifyRDN**. The context flag associated with DCV_DEREF_ALIASES is not relevant to **NWDSModifyRDN** and is ignored.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSModifyDN, **NWDSModifyObject**, **NWDSGetContext**, **NWDSSetContext**

NWDSMoveObject

Moves an NDS object from one container to another and/or renames the object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSMoveObject (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              destParentDN,
   pnstr8              destRDN);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSMoveObject
  (context : NWDSContextHandle;
   objectName : pnstr8;
   destParentDN : pnstr8;
   destRDN : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object to be moved.

destParentDN

(IN) Points to the name of the object's new parent.

destRDN

(IN) Points to the object's new RDN.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSMoveObject can move an object only if it is a leaf object (meaning it does not have any subordinate objects associated with it). However, it may be either an object or its alias.

The new RDN (such as "Hector") may be the same as the original object's RDN or it may be different.

If you are going to rename the object but not move it, you should call **NWDSModifyRDN** instead of **NWDSMoveObject**.

objectName identifies the object whose DN is to be modified. Aliases in the name will not be dereferenced. Aliases are never dereferenced by **NWDSMoveObject**. The setting of the context flag associated with DCV_DEREF_ALIASES is not relevant to **NWDSMoveObject** and is ignored.

destParentDN identifies the name of the parent object the moved object is to be directly subordinate to. The parent object must already exist in the NDS tree.

destRDN specifies the new RDN of the object being moved.

If Hector is represented in the NDS tree as

```
CN=Hector.OU=Graphics.O=WimpleMakers
```

and you want to move Hector to Marketing, for *objectName* pass in and you want to move Hector to Marketing, for *objectName* pass in

```
CN=Hector.OU=Graphics.O=WimpleMakers
```

for *destParentDN* pass in

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

and for *destRDN* pass in

```
CN=Hector
```

On successful completion, Hector is moved to the new location in the NDS tree, and his complete NDS name becomes

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

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSModifyDN, **NWDSSetContext**, **NWDSGetContext**

NWDSOpenConnToNDSServer

Locates a connection to a specific server
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwdsconn.h>

NWDSCCODE N_API NWDSOpenConnToNDSServer (
   NWDSContextHandle   context,
   pnstr8              serverName,
   pNWCONN_HANDLE      connHandle);
```

Pascal Syntax

```
#include <nwdsconn.inc>

Function NWDSOpenConnToNDSServer
  (context : NWDSContextHandle;
   serverName : pnstr8;
   Var connHandle : NWCONN_HANDLE
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

serverName

(IN) Points to the server to receive the request.

connHandle

(OUT) Points to the connection handle.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSOpenConnToNDSServer allows you to locate a connection to a specific server that is specified by an NDS style name. The *serverName* is resolved using NDS and a connection is established, or, if the workstation already has a connection, the connection handle is returned.

NCP Calls

None

See Also

NWDSAuthenticateConn, **NWDSOpenMonitoredConn**

NWDSOpenMonitoredConn

Opens a connection handle to a monitored connection
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSOpenMonitoredConn (
   NWDSContextHandle   context,
   pNWCONN_HANDLE      connHandle);
```

Pascal Syntax

```
#include <nwdsconn.inc>

Function NWDSOpenMonitoredConn
  (context : NWDSContextHandle;
   Var connHandle : NWCONN_HANDLE
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(OUT) Points to the connection handle.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

A monitored connection is set only if **NWDSLogin** has been called. For multiple tree support, the tree name specified in the context handle is used to specify which monitored connection to retrieve. The user is responsible for closing the connection handle.

If the treename is not set in the context, the preferred tree will be used. For requesters that do not support multiple trees, if the treename is specified (not -NULL string) and if the treename is different than the preferred tree, an error will be returned.

NCP Calls

None

See Also

NWDSGetMonitoredConnRef, **NWDSOpenConnToNDSServer**

NWDSOpenStream

Begins access to an attribute of type SYN_STREAM
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSOpenStream (
   NWDSContextHandle     context,
   pnstr8                objectName,
   pnstr8                attrName,
   nflag32               flags,
   NWFILE_HANDLE N_FAR   fileHandle);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSOpenStream
  (context : NWDSContextHandle;
   objectName : pnstr8;
   attrName : pnstr8;
   flags : nflag32;
   Var fileHandle : NWFILE_HANDLE
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object having the attribute that is to be opened.

attrName

(IN) Points to the attribute name whose value is being read.

flags

(IN) Specifies the mode in which the stream is to be opened:

0x00000001L	DS_READ_STREAM
0x00000002L	DS_WRITE_STREAM

fileHandle

(OUT) Points to the file handle appropriate for the platform from which the API is being called.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Before using a SYN_STREAM attribute for the first time, initialize the attribute with a NULL value. This can be done by calling **NWDSPutAttrVal**. If the attribute is not initialized, **NWDSOpenStream** will return ERR_NO_SUCH_VALUE.

All attributes whose syntax is SYN_STREAM must be accessed by first calling **NWDSOpenStream** to retrieve a file handle to be used for accessing the attribute's value. The returned handle is a file handle that is appropriate for the platform on which the application is running. This file handle can be used to access the attribute value through the platform's standard file I/O functions.

Close the file handle by calling the platform's file close function.

You must use the file I/O functions that are appropriate for the platform on which the application is running. For DOS, call **read**, **write**, **close**, and **seek**. For Windows, call **_lread**, **_lwrite**, **_lclose**, and **_llseek**.

Attribute values that are of syntax SYN_STREAM are not accessed by **NWDSGetAttrVal**. When reading the attributes of an object that has a stream attribute (such as Login Script), **NWDSGetAttrVal** returns a zero-length octet string for the value of the stream attribute.

NOTE: For NLM applications, if the handle returned by **NWDSOpenStream** is to be used by **fdopen**, **NWDSOpenStream** must be called with O_TEXT ORed in with the other values in *flags*.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

NWDSPartitionReceiveAllUpdates

Changes the state of the partition so all servers holding a partition replica will send entire partition information to the original partition

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.01, 4.02, 4.1, 4.11

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSPartitionReceiveAllUpdates (
   NWDSContextHandle   context,
   pnstr8              partitionRoot,
   pnstr8              serverName);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSPartitionReceiveAllUpdates
  (context : NWDSContextHandle;
   partitionRoot : pnstr8;
   serverName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

partitionRoot

(IN) Points to the name root object name for the partition.

serverName

(IN) Points to the server name where the partition is located.

Return Values

0x0000	SUCCESSFUL
negative	Negative values indicate errors. See NDS Values (-001 to

negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSPartitionReceiveAllUpdates changes the state of the specified partition to that of a new partition. This results in all servers holding a replica of this partition sending their entire partition information, not just changes, to the partition on the target server.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSPartitionSendAllUpdates

NWDSPartitionSendAllUpdates

Tells the specified partition to send full updates to any server holding a replica of the partition

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.01, 4.02, 4.1, 4.11

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSPartitionSendAllUpdates (
   NWDSContextHandle   context,
   pnstr8              partitionRoot,
   pnstr8              serverName);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSPartitionSendAllUpdates
  (context : NWDSContextHandle;
   partitionRoot : pnstr8;
   serverName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

partitionRoot

(IN) Points to the name root object name for the partition.

serverName

(IN) Points to the server name where the partition is located.

Return Values

0x0000	SUCCESSFUL
negative	Negative values indicate errors. See NDS Values (-001 to

negative value	Negative values indicate errors. See NDS Values (-001 to -699).

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSPartitionReceiveAllUpdates

NWDSPutAttrName

Stores an attribute name in a request buffer to be used by a NDS function
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSPutAttrName (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              attrName);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSPutAttrName
   (context : NWDSContextHandle;
    buf : pBuf_T;
    attrName : pnstr8
) : NWDSCCODE;
```

Parameters

context
(IN) Specifies the NDS context for the request.

buf
(IN) Points to the request buffer in which to store the attribute name.

attrName
(IN) Points to the attribute name to store in the request buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The maximum size of the attribute name is (MAX_DN_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

NCP Calls

None

See Also

NWDSAddObject, **NWDSAddReplica**, **NWDSCompare**, **NWDSPutAttrVal**, **NWDSRead**, **NWDSReadAttrDef**, **NWDSSearch**, **NWDSSplitPartition**

NWDSPutAttrNameAndVal

Stores an attribute name and value in a request buffer to be used by a NDS function

Local Servers: nonblocking

Remote Servers: nonblocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSPutAttrNameAndVal (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              attrName,
   nuint32             syntaxID,
   nptr                attrVal);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSPutAttrNameAndVal (
   context : NWDSContextHandle;
   buf : pBuf_T;
   attrName : pnstr8;
   syntaxID : nuint32;
   attrVal : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the request buffer in which to store the attribute name.

attrName

(IN) Points to the attribute name to store in the request buffer.

syntaxID

(IN) Specifies the data type of the attribute value.

attrVal

(IN) Points to the attribute value to be stored in the request buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSPutAttrNameAndVal combines the functionality of **NWDSPutAttrName** and **NWDSPutAttrVal** and adds error checking so the buffer is always valid.

NWDSPutAttrNameAndVal checks the return values of **NWDSPutAttrName** and **NWDSPutAttrVal**. If either function fails because the buffer was too small, **NWDSPutAttrNameAndVal** will not modify the buffer. Call **NWDSAddObject** or **NWDSModifyObject** to modify the NDS object.

The maximum size of the attribute name is (MAX_DN_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

buf points to a Buf_T, which is allocated by **NWDSAllocBuf** and initialized by **NWDSInitBuf**.

syntaxID tells **NWDSPutAttrNameAndVal** what method to use for converting the attribute value to a machine-transparent form when storing the value in the buffer. Syntax IDs (such as SYN_PATH) are enumerated in NWDSDEFS.H. Syntaxes are described in NDS Attribute Syntax Definitions.

attrVal points to the attribute value to be stored in the request buffer. The type of data pointed to by *attrVal* depends on the indicated attribute syntax. See NDS Attribute Syntax Definitions to determine the data type associated with an attribute.

NCP Calls

None

See Also

NWDSAddObject, **NWDSAddReplica**, **NWDSCompare**,

NWDSModifyObject, **NWDSPutAttrName**, **NWDSPutAttrVal**, **NWDSRead**, **NWDSReadAttrDef**, **NWDSSearch**, **NWDSSplitPartition**

NWDSPutAttrVal

Stores an attribute value in a request buffer to be used by a NDS function

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSPutAttrVal (
   NWDSContextHandle   context,
   pBuf_T              buf,
   nuint32             syntaxID,
   nptr                attrVal);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSPutAttrVal
  (context : NWDSContextHandle;
   buf : pBuf_T;
   syntaxID : nuint32;
   attrVal : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the request buffer being prepared.

syntaxID

(IN) Specifies the data type of the attribute value.

attrVal

(IN) Points to the attribute value to be stored in the request buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

buf points to a Buf_T, which is allocated by **NWDSAllocBuf** and initialized by **NWDSInitBuf**.

The name of the attribute to which the value belongs is specified previously by calling either **NWDSPutChange** or **NWDSPutAttrName** (depending on the nature of the operation).

syntaxID tells **NWDSPutAttrVal** what method to use for converting the attribute value to a machine-transparent form when storing the value in the buffer. Syntax IDs (such as SYN_PATH) are enumerated in NWDSDEFS.H. Syntaxes are described in NDS Attribute Syntax Definitions.

attrVal points to the attribute value to be stored in the request buffer. The type of data pointed to by *attrVal* depends on the indicated attribute syntax. See NDS Attribute Syntax Definitions to determine the data type associated with an attribute.

NCP Calls

None

See Also

NWDSAddObject, **NWDSAddReplica**, **NWDSCompare**, **NWDSModifyObject**, **NWDSPutAttrName**, **NWDSPutChange**, **NWDSSplitPartition**

NWDSPutChange

Stores a change record in a request buffer to be used by **NWDSModifyObject**

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSPutChange (
   NWDSContextHandle   context,
   pBuf_T              buf,
   nuint32             changeType,
   pnstr8              attrName);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSPutChange
  (context : NWDSContextHandle;
   buf : pBuf_T;
   changeType : nuint32;
   attrName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the request buffer where the request will be stored.

changeType

(IN) Specifies the modification type to be performed.

attrName

(IN) Points to the attribute name to be changed.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

A change record includes the name of the attribute and the type of change to be performed.

changeType indicates the type of modification to be performed on an object. The following types of changes are supported:

0	#0 0	DS_ADD_ATTRIBUTE: New attribute to be added to the object (attempting to add an already-existing attribute results in an error)
1	#0 1	DS_REMOVE_ATTRIBUTE: Attribute to be removed from the object (attempting to remove a nonexisting attribute results in an error and removing an attribute is not allowed if the attribute is present in the RDN)
2	#0 2	DS_ADD_VALUE: Values to be added to an attribute (attempting to add an already-existing value or to add a value to a nonexistent attribute results in an error)
3	#0 3	DS_REMOVE_VALUE: Values to be removed from an attribute (an error results if values not in the attribute and removing values is not allowed if any of the values are present in the RDN)
4	#0 4	DS_ADDITIONAL_VALUE: Add an additional value to a multivalued attribute
5	#0 5	DS_OVERWRITE_VALUE: Modify an attribute value without needing to remove the old value first and then add the new value
6	#0 6	DS_CLEAR_ATTRIBUTE: Delete an attribute without checking to see if the attribute exists
7	#0 7	DS_CLEAR_VALUE: Clear an attribute value without checking to see if the value exists

If an attempt is made to modify the Object Class attribute, an error is returned.

A value can be modified by placing a combination of DS_REMOVE_VALUE and DS_ADD_VALUE change records in the same request buffer. This allows the operations to be completed with a single call to **NWDSModifyObject**.

NCP Calls

None

See Also

NWDSPutAttrVal, **NWDSModifyObject**

NWDSPutChangeAndVal

Stores a change record and attribute value in a request buffer to be used by **NWDSModifyObject**

Local Servers: nonblocking

Remote Servers: nonblocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSPutChangeAndVal (
   NWDSContextHandle   context,
   pBuf_T              buf,
   nuint32             changeType,
   pnstr8              attrName,
   nuint32             syntaxID,
   nptr                attrVal);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSPutChangeAndVal (
   context : NWDSContextHandle;
   buf : pBuf_T;
   changeType : nuint32;
   attrName : pnstr8;
   syntaxID : nuint32;
   attrVal : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the request buffer where the request will be stored.

changeType

(IN) Specifies the modification type to be performed.

attrName

(IN) Points to the attribute name to be changed.

syntaxID

(IN) Specifies the data type of the attribute value.

attrVal

(IN) Points to the attribute value to be stored in the request buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSPutChangeAndVal combines the functionality of **NWDSPutChange** and **NWDSPutAttrVal** and adds error checking so the buffer is always valid.

NWDSPutChangeAndVal checks the return values of **NWDSPutChange** and **NWDSPutAttrVal**. If either function fails because the buffer was too small, **NWDSPutChangeAndVal** will not modify the buffer. Call **NWDSModifyObject** to modify the NDS object.

A change record includes the name of the attribute and the type of change to be performed.

changeType indicates the type of modification to be performed on an object. The following types of changes are supported:

0	#00	DS_ADD_ATTRIBUTE: New attribute to be added to the object (attempting to add an already-existing attribute results in an error)
1	#01	DS_REMOVE_ATTRIBUTE: Attribute to be removed from the object (attempting to remove a nonexisting attribute results in an error and removing an attribute is not allowed if the attribute is present in the RDN)
2	#02	DS_ADD_VALUE: Values to be added to an attribute (attempting to add an already-existing value or to add a value to a nonexistent attribute results in an error)
3	#03	DS_REMOVE_VALUE: Values to be removed from an attribute (an error results if values not in the attribute and removing values is not allowed if any of the values are present in the RDN)

4	#0 4	DS_ADDITIONAL_VALUE: Add an additional value to a multivalued attribute
5	#0 5	DS_OVERWRITE_VALUE: Modify an attribute value without needing to remove the old value first and then add the new value
6	#0 6	DS_CLEAR_ATTRIBUTE: Delete an attribute without checking to see if the attribute exists
7	#0 7	DS_CLEAR_VALUE: Clear an attribute value without checking to see if the value exists

If an attempt is made to modify the Object Class attribute, an error is returned.

A value can be modified by placing a combination of DS_REMOVE_VALUE and DS_ADD_VALUE change records in the same request buffer. This allows the operations to be completed with a single call to **NWDSModifyObject**.

buf points to a Buf_T, which is allocated by **NWDSAllocBuf** and initialized by **NWDSInitBuf**.

syntaxID tells **NWDSPutAttrVal** what method to use for converting the attribute value to a machine-transparent form when storing the value in the buffer. Syntax IDs (such as SYN_PATH) are enumerated in NWDSDEFS.H. Syntaxes are described in NDS Attribute Syntax Definitions.

attrVal points to the attribute value to be stored in the request buffer. The type of data pointed to by *attrVal* depends on the indicated attribute syntax. See NDS Attribute Syntax Definitions to determine the data type associated with an attribute.

NCP Calls

None

See Also

NWDSPutAttrVal, **NWDSPutChange**, **NWDSModifyObject**

NWDSPutClassItem

Stores a class name or attribute name in a request buffer to be used by a NDS Schema function

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSPutClassItem (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              itemName);
```

Pascal Syntax

```
#include <nwdsbuft.inc>

Function NWDSPutClassItem
  (context : NWDSContextHandle;
   buf : pBuf_T;
   itemName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the request buffer where the item will be stored.

itemName

(IN) Points to the class name or attribute name to be stored in the request buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Class items are added to one of five class-definition item lists. The first two lists contain the names of classes; the remaining lists contain the names of attributes.

These class-definition item lists must be stored in the request buffer in the following order:

1. Super Class Names
2. Containment Class Names
3. Naming Attribute Names
4. Mandatory Attribute Names
5. Optional Attribute Names

NWDSPutClassItem is used in conjunction with **NWDSBeginClassItem** to add items into the list. **NWDSBeginClassItem** is called to move to the next class-definition item list.

The first time **NWDSBeginClassItem** is called, items added with **NWDSPutClassItem** will be placed in the Super Class Names list.

The second time **NWDSBeginClassItem** is called, items added with **NWDSPutClassItem** will be placed in the Containment Class Names list.

Items are added to the other lists with subsequent calls to **NWDSBeginClassItem** and **NWDSPutClassItem**.

NWDSPutClassItem adds one item each time it is called. To store multiple items in a list, call **NWDSPutClassItem** for each item.

See **Creating a Class Definition** for the complete steps to fill out a buffer to be used for defining a new class.

NCP Calls

None

See Also

NWDSReadClassDef, **NWDSPutClassName**, **NWDSPutSyntaxName**

NWDSPutClassName

Stores a class name in a request buffer to be used by a NDS function
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSPutClassName (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              itemName);
```

Parameters

context

(IN) Indicates the NDS context for the request.

buf

(IN) Points to the request buffer being prepared.

itemName

(IN) Points to the class name to be stored in the request buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSPutClassName is a macro that calls **NWDSPutClassItem**. The **NWDSPutClassName** function makes source code more descriptive by having the function name identify what type of class item is being stored in the request buffer.

Class items are added to one of five class-definition item lists. These class-definition item lists are stored in the buffer in the following order:

1. Super Class Names
2. Containment Class Names
3. Naming Attribute Names
4. Mandatory Attribute Names
5. Optional Attribute Names

The first two lists contain object class names; the remaining lists contain attribute names. **NWDSPutClassName** is used to place class names into the Super Class Names and the Containment Class Names lists. **NWDSPutClassItem** is used for the other lists.

NCP Calls

None

See Also

NWDSReadClassDef, **NWDSPutClassItem**

NWDSPutFilter

Prepares a search filter expression tree in a request buffer so it can be used in a call to **NWDSSearch**

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

NWDSCCODE N_API NWDSPutFilter (
   NWDSContextHandle      context,
   pBuf_T                 buf,
   pFilter_Cursor_T       cur,
   void (N_FAR N_CDECL  *freeVal)(nuint32 syntax, nptr val);
```

Pascal Syntax

```
#include <nwdsfilt.inc>

Function NWDSPutFilter
  (context : NWDSContextHandle;
   buf : pBuf_T;
   cur : pFilter_Cursor_T;
   freeVal : FreeValProc
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the request buffer being prepared.

cur

(IN) Points to a cursor to the filter expression tree being stored in the buffer.

freeVal

(IN) Points to the function used to free the attribute values.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

buf points to a Buf_T, which is allocated by **NWDSAllocBuf** and initialized for a DSV_SEARCH operation by **NWDSInitBuf**.

NWDSPutFilter frees the area allocated to the expression tree, including the area referenced by *cur*. If the application needs to retain the expression tree, the application should save the tree in some form before calling **NWDSPutFilter**.

NOTE: **NWDSPutFilter** always frees the memory allocated to the expression tree, even if **NWDSPutFilter** returns an error. Do not call **NWDSFreeFilter** to free the filter if **NWDSPutFilter** returns an error. Doing so will corrupt memory since the filter will already have been freed.

freeVal points to a function freeing the attribute values. The function is passed the syntax attribute ID and the address of the area to free.*freeVal* can be NULL, in which case no attribute values are freed.

NCP Calls

None

See Also

NWDSAddFilterToken, **NWDSAllocFilter**, **NWDSDelFilterToken**, **NWDSFreeFilter**

NWDSPutSyntaxName

Stores a syntax name in a request buffer to be used by a NDS function
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API NWDSPutSyntaxName (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              syntaxName);
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the request buffer being prepared.

syntaxName

(IN) Points to the syntax name to be stored in the request buffer.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSPutSyntaxName is a macro which calls **NWDSPutClassItem**.

buf points to Buf_T, which is allocated by **NWDSAllocBuf** and initialized by **NWDSInitBuf**.

NCP Calls

None

None

See Also

NWDSReadClassDef, **NWDSPutClassItem**

NWDSRead

Reads values from one or more of the specified object's attributes
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSRead (
   NWDSContextHandle   context,
   pnstr8              object,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint32             iterationHandle,
   pBuf_T              objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSRead
  (context : NWDSContextHandle;
   objectName : pnstr8;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint32;
   objectInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

object

(IN) Points to the name of the object whose information is to be read.

infoType

(IN) Specifies the type of information desired:

0	#0 0	DS_ATTRIBUTE_NAMES
1	#0 1	DS_ATTRIBUTE_VALUES
2	#0 2	DS_EFFECTIVE_PRIVILEGES

allAttrs

(IN) Specifies the scope of the request:

TRUE	Information concerning all attributes is requested
FALSE	Information concerning only attributes named in the *attrNames* parameter is requested

attrNames

(IN) Points to a buffer containing the names of the object's attributes for which information is to be returned.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSRead**.

objectInfo

(OUT) Points to a buffer receiving the requested attribute names and/or values.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

infoType, *allAttrs*, and *attrNames* indicate what attribute information is requested.

If *allAttrs* is TRUE, information about all attributes associated with the object is requested and *attrNames* is ignored (in which case, pass in a NULL for *attrNames*). If *allAttrs* is FALSE, only the attributes specified by the request buffer pointed to by *attrNames* are requested.

If *allAttrs* is FALSE and *attrNames* is NULL, no attribute information is returned; and *infoType* is not meaningful. In this case, the return value of **NWDSRead** can determine whether the specified object exists (verifying the object's distinguished name), or whether access to the object is allowed.

The request buffer pointed to by *attrNames* explicitly specifies the attribute to be returned. If *allAttrs* is TRUE, *attrNames* can be NULL (as mentioned above).

The result buffer pointed to by *objectInfo* received the requested information. This result buffer either contains a list of attribute names or a sequence of attribute-name and attribute-value sets. The type of information returned depends on *infoType* (as mentioned above).

iterationHandle controls retrieval of list results larger than the result buffer pointed to by *attrNames*.

Before the initial call to **NWDSListAttrsEffectiveRights**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when **NWDSRead** returns from its initial call, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSRead** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

NOTE: On large networks, iterative processes, like **NWDSRead**, might take a lot of time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. These processes can be interrupted or aborted using **NWDSCloseIteration**.

Developers should use **NWDSCloseIteration** to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSReadObjectInfo**

NWDSReadAttrDef

Retrieves information about NDS Schema attribute definitions
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdssch.h>

NWDSCCODE N_API NWDSReadAttrDef (
   NWDSContextHandle   context,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint32             iterationHandle,
   pBuf_T              attrDefs);
```

Pascal Syntax

```
#include <nwdssch.inc>

Function NWDSReadAttrDef
  (context : NWDSContextHandle;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint32;
   attrDefs : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

infoType

(IN) Specifies the information type desired (names only, or names and definitions).

allAttrs

(IN) Specifies whether information for all attributes or for selected attributes should be returned.

attrNames

(IN) Points to a request buffer containing the attribute names whose definitions are to be returned.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSReadAttrDef**.

attrDefs

(OUT) Points to a result buffer that receives the requested attribute names and/or definitions.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

infoType, *allAttrs*, and *attrNames* indicate what NDS Schema attribute information is requested.

infoType specifies whether both attribute names and attribute definitions will be retrieved, or only attribute names. The types of choices are the following:

0 DS_ATTR_DEF_NAMES (attribute names only)
1 DS_ATTR_DEFS (attribute names and definitions)

If *allAttrs* is TRUE, information about all attributes in the NDS Schema is requested. In this case, *attrNames* is ignored and can be set to NULL. If *allAttrs* is FALSE, *attrNames* must point to a request buffer containing the attribute names whose information is to be retrieved.

attrNames points to a request buffer containing the attribute names whose information is to be returned.

iterationHandle controls retrieval of results that are larger than the result buffer pointed to by *attrDefs*.

Before the initial call to **NWDSReadAttrDef**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

If, when **NWDSReadAttrDef** returns from its initial call and the result buffer holds the complete results, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSReadAttrDef** in order to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

NOTE: To end the Read operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_READ_ATTR_DEF to free memory and states associated with **NWDSReadAttrDef**.

The level of granularity for partial results is an individual attribute definition.

attrDefs points to a request buffer containing the requested attribute information. This buffer either contains a list of attribute names, or a sequence of attribute names and definitions depending upon the value of *infoType* mentioned above.

Retrieve information about selected attribute definitions using the following steps:

1. Allocate a request buffer by calling **NWDSAllocBuf**. (If you want information about all attributes, you do not need a request buffer and can skip steps 1 through 3.

2. Initialize the request buffer for a DSV_READ_ATTR_DEF operation by calling **NWDSInitBuf**.

3. For each attribute whose information you want to retrieve, store the attribute's name in the request buffer by calling **NWDSPutAttrName**.

4. Allocate a result buffer by calling **NWDSAllocBuf**. (This buffer does not need to be initialized since it is a result buffer.)

5. Set the contents of the iteration handle to NO_MORE_ITERATIONS.

6. Call **NWDSGetAttrDef** with *infoType* set to DS_ATTR_DEF_NAMES to retrieve names only, or with DS_ATRR_DEFS to retrieve names and

attribute definitions. Set *allAttrs* to FALSE if you are using a request buffer, or to TRUE if you are not using a request buffer.

7. Determine the number of attributes whose information is in the result buffer by calling **NWDSGetAttrCount**.
8. For each attribute in the buffer, remove the attribute information by calling **NWDSGetAttrDef**.
9. If the iteration handle is not set to NO_MORE_ITERATIONS, loop to step 6 to retrieve additional attribute information. Otherwise, go to step 10.
10. Free the request buffer by calling **NWDSFreeBuf**.
11. Free the result buffer by calling **NWDSFreeBuf**.

See **NWDSGetAttrDef** to see the type of information returned in the buffer.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSReadClassDef

NWDSReadClassDef

Retrieves information about NDS Schema object class definitions
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdssch.h>

NWDSCCODE N_API NWDSReadClassDef (
   NWDSContextHandle   context,
   nuint32             infoType,
   nbool8              allClasses,
   pBuf_T              classNames,
   pnint32             iterationHandle,
   pBuf_T              classDefs);
```

Pascal Syntax

```
#include <nwdssch.inc>

Function NWDSReadClassDef
  (context : NWDSContextHandle;
   infoType : nuint32;
   allClasses : nbool8;
   classNames : pBuf_T;
   iterationHandle : pnint32;
   classDefs : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

infoType

(IN) Specifies the information type desired (names only, or names and definitions).

allClasses

(IN) Specifies whether information for every object class should be returned.

classNames

(IN) Points to a request buffer containing the names of the object classes whose information is to be returned.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSReadClassDef**.

classDefs

(OUT) Points to a result buffer containing the requested object-class names and/or definitions.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

infoType, *allClasses*, and *classNames* indicate the type of object-class information requested.

infoType specifies whether both class name and class definitions are requested. It also specifies whether to return information about class definitions or expanded class-definitions. The valid values for *infoType* are as follows:

0	DS_CLASS_DEF_NAMES returns class names only.
1	DS_CLASS_DEFS returns both class names and definitions.
2	DS_EXPANDED_CLASS_DEFS returns expanded class definitions.
3	DS_INFO_CLASS_DEFS returns class definitions only.

If *allClasses* is TRUE, information about all classes in the NDS schema is requested and *classNames* is ignored and can be set to NULL. If *allClasses* is FALSE, only the class definitions specified in the request buffer pointed to by *classNames* are requested.

classNames is a request buffer specifying the names of the object-classes whose information you want to return. This can be set to NULL, as mentioned in the previous paragraph.

iterationHandle controls retrieval of results larger than the result buffer pointed to by *classDefs*.

Before the initial call to **NWDSReadClassDef**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

When **NWDSReadClassDef** returns from its initial call, if the result buffer holds the complete results, the location pointed by *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSReadClassDef** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

NOTE: To end the Read operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_READ_CLASS_DEFS to free memory and states associated with **NWDSReadClassDef**.

NOTE: The level of granularity for partial results is an individual class definition. If the buffer is not large enough to hold an entire class definition, ERR_INSUFFICIENT_BUFFER will be returned.

classDefs points to a result buffer containing the requested information. This buffer contains either a list of class names, or a sequence of class names and definitions, or a sequence of class definitions. The type of information returned depends on *infoType*.

Take the following steps to retrieve information about object-class definitions in the NDS Schema

1. Allocate a request buffer by calling **NWDSAllocBuf**. (If you are retrieving information about all class definitions, you do not need a request buffer and can skip steps 1 through 3.)

2. Initialize the request buffer for a DSV_READ_CLASS_DEF operation by calling **NWDSInitBuf**.

3. For each object class you want to receive information about, store the object-class name in the request buffer by calling **NWDSPutClassName**
.

4. Allocate a result buffer by calling **NWDSAllocBuf**. (It does not need to

be initialized since it is a result buffer.)

5. Set the contents of the iteration handle to NO_MORE_ITERATIONS.

6. Retrieve the object-class information by calling **NWDSReadClassDef**.

7. Determine the number of object classes whose information is in the result buffer by calling **NWDSGetClassDefCount**.

8. If you have chosen object names and definitions, retrieve each object-class definition from the buffer by using the steps in the list following this list.

9. If the iteration handle is not set to NO_MORE_ITERATIONS, loop to step 6 to retrieve more information about object classes. Otherwise, go to step 10.

10. Free the request buffer by calling **NWDSFreeBuf**.

11. Free the result buffer by calling **NWDSFreeBuf**.

For each object-class definition in the buffer, take the following steps to remove the information:

1. Read the name (and other information) of the current object class whose definition is in the buffer by calling **NWDSGetClassDef**. (If you called **NWDSReadClassDef** with *infoType* being DS_INFO_CLASS_DEFS, skip the rest of the following steps and call **NWDSGetClassItem**. **NWDSGetClassDef** contains an extra parameter that is not used if the *infoType* parameter is set to DS_INFO_CLASS_DEFS.)

2. Move to the result buffer's Super Class Names list and determine the number of super-class names in the list by calling **NWDSGetClassItemCount**. (Moving to the list and determining the number of class names is accomplished with one call to **NWDSGetClassItemCount**.

3. For each super-class name in the Super Class Names list, retrieve the super-class name by calling **NWDSGetClassItem**.

4. Move to the result buffer's Containment Class Names list and determine the number of containment-class names in the list by calling **NWDSGetClassItemCount**.

5. For each containment class name in the Containment Class Names list, retrieve the containment-class name by calling **NWDSGetClassItem**.

6. Move to the result buffer's Naming Attribute Names list and determine the number of naming-attribute names in the list by calling **NWDSGetClassItemCount**.

7. For each naming-attribute name in the Naming Attribute Names list, retrieve the naming-attribute name by calling **NWDSGetClassItem**.

8. Move to the result buffer's Mandatory Attribute Names list and determine the number of mandatory-attribute names in the list by calling **NWDSGetClassItemCount**.

9. For each mandatory-attribute name in the Mandatory Attribute Names list, retrieve the naming attribute name by calling **NWDSGetClassItem**.

10. Move to the result buffer's Optional Attribute Names list and determine the number of optional-attribute names in the list by calling **NWDSGetClassItemCount**.

11. For each optional-attribute name in the Optional Attribute Names list, retrieve the optional-attribute name by calling **NWDSGetClassItem**.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSReadAttrDef

NWDSReadObjectDSIInfo

Returns DSI object information not stored in the attributes of an object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSReadObjectDSIInfo (
   NWDSContextHandle   context,
   pnstr8              object,
   nuint32             infoLength,
   nptr                 objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSReadObjectDSIInfo (
   context : NWDSContextHandle;
   object1 : pnstr8;
   infoLength : nuint32;
   objectInfo : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

object

(IN) Points to the name of the object for which information is to be returned.

infoLength

(IN) Specifies the size of the *objectInfo* buffer.

objectInfo

(OUT) Points to the non-attribute information about the object.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Object information can be useful to applications browsing the NDS tree.

If aliases are to be dereferenced (the context flag associated with DCV_DEREF_ALIASES is set) and *object* passes an alias name for the object, the name pointed to by *infoLength* is the dereferenced name of the object.

The caller must allocate sufficient memory to contain the data elements specified by the DSI Flags.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSGetDSIInfo, **NWDSGetObjectNameAndInfo**, **NWDSRead**, **NWDSReadObjectInfo**

NWDSReadObjectInfo

Returns object information not stored in the object's attributes
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSReadObjectInfo (
   NWDSContextHandle   context,
   pnstr8              object,
   pnstr8              distinguishedName,
   pObject_Info_T      objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSReadObjectInfo
  (context : NWDSContextHandle;
   objectName : pnstr8;
   distinguishedName : pnstr8;
   objectInfo : pObject_Info_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

object

(IN) Points to the name of the object for which information is to be returned.

distinguishedName

(OUT) Points to the object name.

objectInfo

(OUT) Points to the non-attribute information about the object.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Object information can be useful to applications browsing the NDS tree.

See DSI Flags.

If aliases are to be dereferenced (the context flag associated with DSV_DEREF_ALIASES is set) and *object* passes an alias name for the object, the name pointed to by *distinguishedName* is the dereferenced name of the object.

The caller must allocate memory to hold the distinguished name of the object. The size of the memory is (MAX_DN_CHARS+1)*sizeof(character size), where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double byte). One character is used for NULL termination.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRead

NWDSReadReferences

Searches all of the replicas on a particular server and returns any objects that contain attributes that reference the specified object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSReadReferences (
   NWDSContextHandle   context,
   pnstr8              serverName,
   pnstr8              objectName,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   uint32              timeFilter,
   pnint32             iterationHandle,
   pBuf_T              objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSReadReferences
  (context : NWDSContextHandle;
   serverName : pnstr8;
   objectName : pnstr8;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   timeFilter : nuint32;
   iterationHandle : pnint32;
   objectInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

serverName

(IN) Points to the server name to read from.

objectName

(IN) Points to the object name whose information is to be read.

infoType

(IN) Specifies the type of information to be returned.

allAttrs

(IN) Specifies the scope of the request.

attrNames

(IN) Points to a request buffer containing the names of attributes whose information is to be returned. This can be NULL.

timeFilter

(IN) Specifies the attribute modification time to be used as a filter. This parameter must be specified and cannot be NULL.

iterationHandle

(IN/OUT) Points to information necessary to resume subsequent calls to **NWDSReadReferences**. This should be initially set to NO_MORE_ITERATIONS.

objectInfo

(OUT) Points to a result buffer containing the requested attribute names or attribute names and values. This parameter can be NULL.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0xFE0D	UNI_NO_OFFSET_DEFAULT
0xFE0F	UNI_HANDLE_MISMATCH
0xFE10	UNI_HANDLE_BAD
0xFEB5	ERR_NULL_POINTER
0xFEBB	ERR_INVALID_ATTR_SYNTAX
0xFEBD	ERR_BUFFER_ZERO_LENGTH
0xFEBE	ERR_INVALID_HANDLE
0xFEBF	ERR_UNABLE_TO_MATCH
0xFED1	ERR_BAD_CONTEXT
0xFED3	ERR_NOT_ENOUGH_MEMORY
negative	Negative values indicate errors. See NDS Values (-001 to

value	-699).

Remarks

infoType, *allAttrs*, and *attrNames* indicate what attribute information is requested.

infoType specifies whether both attribute names and values are requested:

0	DS_ATTRIBUTE_NAME defines attribute names only
1	DS_ATTRIBUTE_VALUES defines both attribute names and values

If *allAttrs* is TRUE, information about all attributes associated with the object is requested and *attrNames* is ignored (in which case, assign a NULL pointer to *attrNames*). If *allAttrs* is FALSE, only the attributes specified by the request buffer pointed to by *attrNames* are requested.

If *allAttrs* is FALSE and *attrNames* is NULL, no attribute information is returned. *infoType* is not meaningful. In this case, the return value of **NWDSReadReferences** can determine whether the specified object exists (verifying the objects distinguished name), or whether access to the object is allowed.

The request buffer pointed to by *attrNames* explicitly specifies the attributes to be returned. For more information, see Preparing NDS Output Buffers.

This result buffer either contains a list of attribute names or a sequence of attribute name and value sets. The type of information returned depends on *infoType*. For more information, see Retrieving Results from NDS Output Buffers.

iterationHandle controls retrieval of search results larger than the result buffer pointed to by *objectInfo*.

Before the initial call to **NWDSReadReferences**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

When **NWDSReadReferences** returns from its initial call, if the result buffer holds the complete results, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSReadReferences** to obtain further portions of the results. When the results are completely retrieved, *iterationHandle* will be set to NO_MORE_ITERATIONS.

To end the Read operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_READ_REFERENCES to free memory and states associated with the Read operation.

The level of granularity for partial results is an individual value of an

The level of granularity for partial results is an individual value of an attribute. If an attribute is multivalued and its values are split across two or more **NWDSReadReferences** results, the attribute name is repeated in each result.

The results of **NWDSReadReferences** are not ordered and might not be returned in alphabetical order.

If *allAttrs* is set to DS_ATTRIBUTE_VALUES, specifying the Read operation should return both attribute names and values, you cannot remove only the names from the result buffer. You must remove the information in the correct order of the attribute name first, then all of the values associated with the attribute. Then remove the next attribute name and its values. Otherwise, **NWDSGetAttrName** will return erroneous information.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSReadObjectInfo

NWDSReadSyntaxDef

Returns the syntax definition for a given NDS syntax identifier

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSReadSyntaxDef (
   NWDSContextHandle   context,
   nuint32             syntaxID,
   pSyntax_Info_T      syntaxDef);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSReadSyntaxDef
  (context : NWDSContextHandle;
   syntaxID : nuint32;
   syntaxDef : pSyntax_Info_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

syntaxID

(IN) Specifies the syntax identifier whose definition is to be returned.

syntaxDef

(OUT) Points to a Syntax_Info_T structure, which receives the syntax definition.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

syntaxID is an identifier (not a string) of a syntax. These identifiers (such as SYN_TEL_NUMBER) are enumerated in NWDSDEFS.H.

This is a local function. Syntaxes are well known.

NCP Calls

None

NWDSReadSyntaxes

Enumerates syntax definitions or retrieves specific NDS Schema syntax definitions

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSReadSyntaxes (
   NWDSContextHandle   context,
   nuint32             infoType,
   nbool8              allSyntaxes,
   pBuf_T              syntaxNames,
   pnint32             iterationHandle,
   pBuf_T              syntaxDefs);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSReadSyntaxes
  (context : NWDSContextHandle;
   infoType : nuint32;
   allSyntaxes : nbool8;
   syntaxNames : pBuf_T;
   iterationHandle : pnint32;
   syntaxDefs : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

infoType

(IN) Specifies the type of information desired (names only, or names and syntax definitions).

allSyntaxes

(IN) Specifies the scope of the request (all or selected syntaxes).

syntaxNames

(IN) Points to a request buffer containing the names of the syntaxes for which information is to be returned.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSReadSyntaxes**.

syntaxDefs

(OUT) Points to a result buffer containing the requested syntax names and/or definitions.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

infoType, *allSyntaxes*, and *syntaxNames* indicate what syntax information is requested.

infoType specifies whether both syntax names and syntax values are requested. The valid values for *infoType* are as follows:

0	DS_SYNTAX_NAMES returns syntax names only
1	DS_SYNTAX_DEFS returns syntax names and values

If *allSyntaxes* is TRUE, information about all syntaxes associated with the object is requested and *syntaxNames* is ignored). If *allSyntaxes* is FALSE, only the syntaxes specified by *syntaxNames* are requested.

If *allSyntaxes* is FALSE and *syntaxNames* is NULL, no syntax information is returned. *infoType* is not meaningful.

syntaxNames is a request buffer containing the names of the specific syntaxes whose information is to be returned. It is used to explicitly specify the syntaxes to be returned.

iterationHandle controls the retrieval of results that are larger than the result buffer pointed to by *syntaxDefs*.

Before the initial call to **NWDSReadSyntaxes**, set the contents of the iteration handle pointed to by *iterationHandle* to NO_MORE_ITERATIONS.

When **NWDSReadSyntaxes** returns from its initial call, if the result buffer holds the complete results, the location pointed to by *iterationHandle* is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSReadSyntaxes** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

NOTE: To end the Read operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_READ_SYNTAXES to free memory and states associated with **NWDSReadSyntaxes**.

The level of granularity for partial results is an individual syntax value of a value set. If a value set is split across two or more **NWDSReadSyntaxes** results, the syntax name of the current value set is repeated in each result.

syntaxDefs points to a result buffer receiving the requested information. This buffer contains either a list of syntax names or a sequence of syntax name and value sets, depending upon the value of *infoType*.

Read results from the buffer by calling **NWDSGetSyntaxCount** and **NWDSGetSyntaxDef**.

The results of **NWDSReadSyntaxes** are not ordered, meaning the syntaxes might not be stored in the result buffer in alphabetical order.

For more information, see Retrieving Syntax Names and Definitions.

NCP Calls

None

NWDSReloadDS

Requests a replica to synchronize with a specific server
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSReloadDS (
   NWDSContextHandle   context,
   pnstr8              serverName);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSReloadDS
  (context : NWDSContextHandle;
   serverName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

serverName

(IN) Specifies the server to send the request to.

Return Values

0x0000	SUCCESSFUL
nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSReloadDS requests the server to reload the DS.NLM.

NCP Calls

None

NWDSRemoveAllTypes

Removes all attribute types from a distinguished name
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsname.h>

NWDSCCODE N_API NWDSRemoveAllTypes (
   NWDSContextHandle   context,
   pnstr8              name,
   pnstr8              typelessName);
```

Pascal Syntax

```
#include <nwdsname.inc>

Function NWDSRemoveAllTypes
  (context : NWDSContextHandle;
   name : pnstr8;
   typelessName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

name

(IN) Points to the object name.

typelessName

(OUT) Points to the object name with the attribute types removed.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSRemoveAllTypes takes the typed name

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

and returns the untyped name

```
Bob.Marketing.WimpleMakers
```

Removal of types is not done relative to the current name context. Therefore, it is not guaranteed that **NWDSCanonicalizeName** can restore the correct types.

The caller must allocate the memory pointed to by *typelessName*. The size of the memory is (MAX_DN_CHARS+1)*sizeof(character size), where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double byte). One character is used for NULL termination.

If the name is already untyped, the same untyped name will be returned.

NCP Calls

None

NWDSRemoveAttrDef

Deletes an attribute definition from the NDS Schema

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdssch.h>

NWDSCCODE N_API NWDSRemoveAttrDef (
   NWDSContextHandle   context,
   pnstr8              attrName);
```

Pascal Syntax

```
#include <nwdssch.inc>

Function NWDSRemoveAttrDef
  (context : NWDSContextHandle;
   attrName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

attrName

(IN) Points to the name of the attribute definition to be removed.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS

0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

An attribute definition can be deleted only if it is not in use in any object class definition, and only if the attribute definition is not flagged as used by the name server.

attrName identifies the attribute definition to be deleted from the Schema.

NOTE: Clients cannot subtract from the standard set of attribute definitions defined by the NDS Base Schema (these attributes are flagged nonremovable). Clients can, however, add and remove non-standard definitions (if not in use).

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRemoveClassDef

NWDSRemoveClassDef

Deletes a class definition from the NDS Schema
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdssch.h>

NWDSCCODE N_API NWDSRemoveClassDef (
   NWDSContextHandle   context,
   pnstr8              className);
```

Pascal Syntax

```
#include <nwdssch.inc>

Function NWDSRemoveClassDef
  (context : NWDSContextHandle;
   className : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

className

(IN) Points to the class name to be removed.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Calling **NWDSRemoveClassDef** is not allowed if the class is referenced by any other class, or if objects of this class exist in NDS.

className identifies the class whose definition is to be removed.

NOTE: Clients cannot subtract from the standard set of class definitions defined by the NDS Base Schema (these are flagged nonremovable). Clients can, however, add and remove non-standard definitions (if not in use).

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRemoveAttrDef

NWDSRemoveObject

Removes a leaf object (either an object or an alias) from the NDS tree
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSRemoveObject (
   NWDSContextHandle   context,
   pnstr8              object);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSRemoveObject
  (context : NWDSContextHandle;
   objectName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

object

(IN) Points to the name of the object to be removed.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS

0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Aliases are never dereferenced by **NWDSRemoveObject**. The setting of the context flag associated with DCV_DEREF_ALIASES is not relevant and is ignored.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

NWDSRemovePartition

Removes an existing partition from NDS by deleting its master replica
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSRemovePartition (
   NWDSContextHandle   context,
   pnstr8              partitionRoot);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSRemovePartition
  (context : NWDSContextHandle;
   partitionRoot : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

partitionRoot

(IN) Points to the name of the root object of the partition to be removed.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS

0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The partition must be completely empty (except for the root object) or the deletion will fail. In addition, no other replicas can exist.

Remove other replicas of the partition beforehand by calling **NWDSRemoveReplica**.

partitionRoot points to the name of the root object in the partition. Since **NWDSRemovePartition** must be performed on the partition's master replica, it is assumed the operation will be performed on the server storing this replica.

Aliases are never dereferenced by **NWDSRemovePartition**. The setting of the NDS context flag associated with DCV_DEREF_ALIASES is not relevant and is ignored.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRemoveReplica

NWDSRemoveReplica

Removes a replica from the replica set of an NDS partition
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSRemoveReplica (
   NWDSContextHandle   context,
   pnstr8              server,
   pnstr8              partitionRoot);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSRemoveReplica
  (context : NWDSContextHandle;
   server : pnstr8;
   partitionRoot : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

server

(IN) Points to the server name where the replica is stored.

partitionRoot

(IN) Points to the name of the root object of the NDS partition whose replica is being deleted.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FF	Failure not related to NDS
0x89FE	BAD_PACKET
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSRemoveReplica removes any replica except the master replica of a partition.

Remove the master replica by calling **NWDSRemovePartition** after all other replicas have been removed by calling **NWDSRemoveReplica**.

Aliases are never dereferenced by **NWDSRemoveReplica**. The setting of the NDS Context flag associated with DCV_DEREF_ALIASES is not relevant and is ignored.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSRemovePartition

NWDSRemSecurityEquiv

Removes a security equivalence from the specified object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE NWDSRemSecurityEquiv (
   NWDSContextHandle   context,
   pnstr8             *equalFrom,
   pnstr8             *equalTo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSRemSecurityEquiv
  (context : NWDSContextHandle;
   equalFrom : pnstr8;
   equalTo : pnstr8
) : NWDSCCODE;
```

Parameters

context
(IN) Specifies the NDS context for the request.

equalFrom
(IN) Points to the name of the object whose Security Equivalence attribute is to be modified.

equalTo
(IN) Points to the object name to be removed from the Security Equivalence attribute of the object specified by *equalFrom*.

Return Values

0x0000	SUCCESSFUL

negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

If **NWDSRemSecurityEquiv** is successful, it will remove the name of the object specified by *equalTo* from the Security Equals attribute of the object specified by *equalFrom*. (Security Equals is a multivalued attribute.)

If the object specified by *equalFrom* does not contain sufficient rights to remove the security equivalence to its list, **NWDSAddSecurityEquiv** will return ERR_NO_ACCESS.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAddSecurityEquiv

NWDSRepairTimeStamps

Sets the time stamps for all of a partition's objects and object attributes to the current time on the NetWare server where the master replica is located

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSRepairTimeStamps (
   NWDSContextHandle   context,
   pnstr8              partitionRoot);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSRepairTimeStamps
  (context : NWDSContextHandle;
   partitionRoot : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

partitionRoot

(IN) Points to the name of the partition's root object.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS

0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSRepairTimeStamps sets the time stamps on all of the objects and their attributes, even if valid time stamps exist. It will replace information such as the creation dates of the attributes.

After **NWDSRepairTimeStamps** is called, the NDS will synchronize all replicas to match the information in the master replica.

CAUTION: Because of the wide scope of changes made by NWDSRepairTimeStamps, it should be used only to recover from a catastrophic failure.

One concern with using NWDSRepairTimeStamps is that it can result in the loss of information. For example, any changes that have been made on replicas other than the master replica will be lost if they have not been synchronized with the master replica before NWDSRepairTimeStamps is called.

Another concern is with applications that use NDS Event Notification Services. After NWDSRepairTimeStamps is called, NDS will produce event notifications for every object and attribute on the master replica. It will also provide the same notification each time one of the other replicas is synchronized with the master replica.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

NWDSReplaceAttrNameAbbrev

Replaces the abbreviated attribute name with its unabbreviated name
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSReplaceAttrNameAbbrev (
   NWDSContextHandle   context,
   pnstr8              inStr,
   pnstr8              outStr);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSReplaceAttrNameAbbrev
  (context : NWDSContextHandle;
   inStr : pnstr8;
   outStr : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context of the request.

inStr

(IN) Points to *attrName*.

outStr

(OUT) Points to the long form of the attribute name.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSReadClassDef returns the abbreviated form of some of the common naming attributes (CN, C, O, OU, L, S, and SA). The long form of these attributes (Common Name, Country Name, Organization Name, Organizational Unit Name, and so on) is returned from **NWDSReplaceAttrNameAbbrev** and pointed to by *outStr*.

The user must allocate space for the long form of the attribute name. The size of the allocated memory is

```
((MAX_SCHEMA_NAME_CHARS)+1)*sizeof(character size)
```

where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

If the name pointed to by *inStr* is not an abbreviated name, the contents of *inStr* will be copied to *outStr*.

NCP Calls

None

NWDSResetNDSStatistics

Retrieves the NDS statistics
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSResetNDSStatistics (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSResetNDSStatistics
  (context : NWDSContextHandle;
   serverName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

serverName

(IN) Specifies the server to send the request to.

Return Values

0x0000	SUCCESSFUL
nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSResetNDSStatistics clears the statistics (resets the counters) that can be reported by **NWDSGetNDSStatistics**.

NCP Calls

None

See Also

NWDSGetNDSStatistics

NWDSResolveName

Returns a connection handle and an object ID for the object name
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsname.h>

NWDSCCODE N_API NWDSResolveName (
   NWDSContextHandle     context,
   pnstr8                objectName,
   NWCONN_HANDLE N_FAR  *conn,
   pnuint32              objectID);
```

Pascal Syntax

```
#include <nwdsname.inc>


Function NWDSResolveName
  (context : NWDSContextHandle;
   objectName : pnstr8;
   Var conn : NWCONN_HANDLE;
   objectID : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object to get the ID for.

conn

(OUT) Points to the connection handle where the object resides.

objectID

(OUT) Points to the NDS object ID.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The returned connection handle is the NetWare server where the object is stored.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAddObject, **NWDSAuditGetObjectID**

NWDSRestoreObject

Restores an object's attribute names and values that were retrieved by calling **NWDSBackupObject**

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSRestoreObject (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnint32             iterationHandle,
   nbool8              more,
   nuint32             size,
   pnuint8             objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSRestoreObject
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint32;
   more : nbool8;
   size : nuint32;
   objectInfo : pnuint8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the object name for which information is to be returned.

more

(IN) Specifies a partial message.

size

(IN) Specifies the length of the information to be restored.

objectInfo

(IN) Points to the starting location of the information to be restored.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSRestoreObject is used to restore the attributes and attribute values for one object at a time. To restore the entire directory, **NWDSRestoreObject** must be called for each object that is to be restored.

iterationHandle and *more* are used to control the restoring of an object's information which cannot be restored with one call to **NWDSRestoreObject**. If more information will be coming after the current call, *more* should be set to TRUE. Otherwise, it should be set to FALSE. In the initial call to **NWDSRestoreObject**, *iterationHandle* should point to NWDS_ITERATION which has been set to NO_MORE_ITERATIONS.

After calling **NWDSRestoreObject** for the last time, and setting *more* to FALSE, the value pointed to by *iterationHandle* will be set to NO_MORE_ITERATIONS on return.

size specifies the length of the information pointed to by *objectInfo*. This is the information saved after calling **NWDSBackupObject**.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSBackupObject

NWDSReturnBlockOfAvailableTrees

Scans the bindery of the specified connection and returns matching tree objects

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsconn.h>


N_EXTERN_LIBRARY (NWDSCCODE)NWDSReturnBlockOfAvailableTrees (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   pnstr               scanFilter,
   pnstr               lastBlocksString,
   pnstr               endBoundString,
   nuint32             maxTreeNames,
   ppnstr              arrayOfNames,
   pnuint32            numberOfTrees,
   pnuint32            totalUniqueTrees);
```

Pascal Syntax

```
#include <nwdsconn.inc>

Function NWDSReturnBlockOfAvailableTrees
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   scanFilter : pnstr;
   lastBlocksString : pnstr;
   endBoundString : pnstr;
   maxTreeNames : nuint32;
   Var arrayOfNames : pnstr;
   Var numberOfTrees : nuint32;
   Var totalUniqueTrees : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(IN) Specifies the connection handle to be used in scanning for NDS trees.

scanFilter

(IN) Points to an ASCII string that defines the scan filter (can contain wildcards).

lastBlocksString

(IN) Points to the last tree name that was returned during a previous scan (used to continue scanning for more names with the same scan filter or pass NULL).

endBoundString

(IN) Points to a string (used in conjunction with the *scanFilter* parameter) that sets up a range of tree names to scan (can contain wildcards).

maxTreeNames

(IN) Maximum number of tree names to return (size of the *arrayOfNames* buffer).

arrayOfNames

(OUT) Points to the first element of an output buffer that will be used to return the tree names.

numberOfTrees

(OUT) Actual number of tree names that were returned.

totalUniqueTrees

(OUT) The total number of tree names found that match the scan criteria. This number might be greater than *numberOfTrees*.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
	NO_SUCH_OBJECT
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Setting Up the Scan

To set up a scan, place a filter string in scanFilter. The filter string can contain a wildcard, such as "nov*". The results of this scan would include all tree names that begin with "nov".

all tree names that begin with "nov".

If you want to set an end boundary for the scan, place a second filter string in *endBoundString*. For example, if *scanFilter* contains "a*" and *scanFilter* contains "ac*", the scan results would include all tree names that begin with "a" that are less than the ordinal value of "ad'.

Initializing the Output Buffer

The caller is responsible for setting up and initializing an output buffer before calling this function. The caller must supply the array of pointers as well as supplying strings of NW_MAX_TREE_NAME_LEN for each element in the array. The following example demonstrates initializing the buffer.

```
main()
{
   ppnstr8   names;
   int       i;
   int       blockOfTreesCount = 25;

      names = malloc(sizeof(pnstr8) * blockOfTreesCount);

      for (i=0; i<25; i++)
      {
         names[i] = malloc(NW_MAX_TREE_NAME_LEN);
      }

      NWDSReturnBlockOfAvailableTrees(,,,,,,names,,);
}
```

The caller could set *maxTreeNames* equal to 0 and call **NWDSReturnBlockOfAvailableTrees**. The value returned in *totalUniqueTrees* could then be used to determine how much memory to allocate for *arrayOfNames*.

NOTE: NW_MAX_TREE_NAME_LEN contains the maximum length of non-unicode names, and NW_MAX_TREE_NAME_BYTES contains the maximum length of unicode names.

Continuing a Scan

If the value returned in *totalUniqueTrees* is greater than the value returned in *numberOfTrees*, there are more tree names that meet the scan filter criteria than were returned in *arrayOfNames*. If this is the case, the caller could make a subsequent call to **NWDSReturnBlockOfAvailableTrees** and begin the scan where the previous call left off.

To set up a subsequent call, keep the values of *scanFilter* and *endBoundString* the same as before, and place the name that was returned in the last element of *arrayOfNames* into *lastBlocksString*.

NCP Calls

NCP Calls

0x2222 23 55   Get Server Sources Information

See Also

NWDSScanConnsForTrees, **NWDSScanForAvailableTrees**

NWDSScanConnsForTrees

Scans existing connections for tree names

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSScanConnsForTrees (
   NWDSContextHandle   context,
   nuint               numOfPtrs,
   pnuint              numOfTrees,
   pnstr8              treeBufPtrs);
```

Pascal Syntax

```
#include <nwdsconn.inc>

Function NWDSScanConnsForTrees
  (context : NWDSContextHandle;
   numOfPtrs : nuint;
   Var numOfTrees : nuint;
   Var treeBufPtrs : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

numOfPtrs

(IN) Specifies the number of pointers available in *treeBufPtrs*.

numOfTrees

(OUT) Points to the number of tree names that can be returned by **NWDSScanConnsForTrees**.

treeBufPtrs

(OUT) Points to an array of pointers that will receive the tree names.

Return Values

0x0000	SUCCESSFUL

negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSScanConnsForTrees scans existing connections and returns a list of tree names associated with those connections. The list does not include duplicates and is sorted by the defined collation table.

The *numOfPtrs* parameter indicates the maximum number of strings that may be assigned by **NWDSScanConnsForTrees**. The *numOfTrees* parameter specifies the number of strings assigned to *treeBufPtrs*. In the event that *numOfTrees* exceeds *numOfPtrs*, *numOfPtrs* strings will be assigned and *numOfTrees* will be returned.

The maximum tree name length is specified by NW_MAX_TREE_NAME_LEN, which is a constant defined to be 33 bytes in length. Unicode defines NW_MAX_TREE_NAME_BYTES to be 66 bytes in length.

The tree names returned imply authentication since a connection isn't designated as Bindery or NDS until authentication.

The *context* parameter is used to determine the character type for the tree name (that is, local code page or Unicode).

NOTE: When **NWDSScanConnsForTrees** is called on a workstation running the new Client32, which currently runs on MS Windows95, it does not actually scan connections. This call utilizes Client32's ability to scan for identities.

NCP Calls

None

See Also

NWDSScanForAvailableTrees, **NWDSReturnBlockOfAvailableTrees**

NWDSScanForAvailableTrees

Scans a connection for tree objects

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSScanForAvailableTrees (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   pnstr               scanFilter,
   pnint32             scanIndex,
   pnstr               treeName);
```

Pascal Syntax

```
#include <nwdsconn.inc>

Function NWDSScanForAvailableTrees
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   scanFilter : pnstr;
   Var scanIndex : nint32;
   treeName : pnstr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(IN) Specifies the connection handle to be used in scanning for NDS trees.

scanFilter

(IN) Points to an ASCII string that allows wildcards to be specified in the scan.

scanIndex

(IN/OUT) Points to the index to be used on the next iteration of the scan.

treeName

(OUT) Points to the name of the tree found in the scan operation.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSScanForAvailableTrees uses the connection specified in *connHandle* to scan for NDS tree objects (object type 0x7802) using the server bindery (the dynamic bindery is used for NetWare 4 server).

The *scanFilter* value allows wildcard matching to be specified for the scan operation. The *scanIndex* value should be initially set to -1 and must not be altered by the user after the first call.

Unlike other NDS functions, there is no need to call **NWDSCloseIteration** to discontinue calling **NWDSScanForAvailableTrees** once the search is begun.

The *context* parameter is used to determine the character type for the tree name (that is, local code page or Unicode).

NCP Calls

0x2222 23 55	Scan Bindery Object

See Also

NWDSScanConnsForTrees, **NWDSReturnBlockOfAvailableTrees**

NWDSSearch

Searches a region of NDS for objects satisfying a specified set of requirements

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSSearch (
   NWDSContextHandle   context,
   pnstr8              baseObjectName,
   nint                scope,
   nbool8              searchAliases,
   pBuf_T              filter,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint32             iterationHandle,
   nint32              countObjectsToSearch,
   pnint32             countObjectsSearched,
   pBuf_T              objectInfo);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSSearch
  (context : NWDSContextHandle;
   baseObjectName : pnstr8;
   scope : nint;
   searchAliases : nbool8;
   filter : pBuf_T;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint32;
   countObjectsToSearch : nint32;
   countObjectsSearched : pnint32;
   objectInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

baseObjectName

(IN) Points to the name of a subtree root to be searched.

scope

(IN) Specifies the depth of the search.

searchAliases

(IN) Specifies whether to dereference aliases in the search subtree:

TRUE	Aliases will be dereferenced
FALSE	Aliases will not be dereferenced

filter

(IN) Points to a search filter constructed by calling the **NWDSAddFilterToken** function.

infoType

(IN) Specifies the type of information to be returned:

TRUE	Specifies attribute names and values
FALSE	Specifies attribute names only

allAttrs

(IN) Specifies the scope of the request:

TRUE	Return information concerning all attributes
FALSE	Return information for only the attributes named in the *attrNames* parameter

attrNames

(IN) Points to the names of the attributes for which information is to be returned.

iterationHandle

(IN/OUT) Points to information needed to resume subsequent iterations of **NWDSSearch**.

countObjectsToSearch

(IN) Is reserved.

countObjectsSearched

(OUT) Points to the number of objects searched by the server.

objectInfo

(OUT) Points to an output buffer containing the names of the objects along with any requested attribute values satisfying the search.

Return Values

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSSearch succeeds if the base object is located, regardless of whether there are any subordinates to the base object.

If a replica-specific error is returned, **NWDSSearch** will try an alternate replica for the partition.

The *baseObjectName* parameter identifies the object (or possibly the root) to which the search is relative. If the string is empty, the current context is selected as the base object.

The *scope* parameter can have the following values:

0 DS_SEARCH_ENTRY Search applies only to the base object.

1 DS_SEARCH_SUBORDINATES Search only applies to the immediate subordinates of the base object.

2 DS_SEARCH_SUBTREE Search applies to the base object and all of its subordinates.

Aliases are dereferenced while locating the base object unless the context flag associated with DCV_DEREF_ALIASES is not set. Aliases among the subordinates of the base object are dereferenced during the search unless the *searchAliases* parameter is FALSE. If the *searchAliases* parameter is TRUE, the search continues in the subtree of the aliased object.

The *filter* parameter eliminates objects not of interest to the application. Information is returned only on objects that satisfy the filter.

The *infoType*, *allAttrs*, and *attrNames* parameters indicate what attribute information is requested.

If the *allAttrs* parameter is TRUE, information about all attributes associated with the object is requested and the *attrNames* parameter is ignored (in which case, the *attrNames* parameter can be NULL). If the *allAttrs* parameter is FALSE, only the attributes specified by the *attrNames* parameter are requested.

If the *allAttrs* parameter is FALSE and the *attrNames* parameter is NULL, no attribute information is returned and the *infoType* parameter is not meaningful. In this case, the value returned by **NWDSSearch** determines whether the specified object exists, or whether access to the object is allowed.

The *iterationHandle* parameter controls the retrieval results that are larger than the result buffer pointed to by the *objectInfo* parameter.

Before calling **NWDSSearch** initially, set the contents of the iteration handle pointed to by the *iterationHandle* parameter to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when **NWDSSearch** returns from its initial call, the location pointed to by the *iterationHandle* parameter is set to NO_MORE_ITERATIONS. If the iteration handle is not set to NO_MORE_ITERATIONS, use the iteration handle for subsequent calls to **NWDSSearch** to obtain further portions of the results. When the results are completely retrieved, the contents of the iteration handle will be set to NO_MORE_ITERATIONS.

To end the Search operation before the complete results have been retrieved, call **NWDSCloseIteration** with a value of DSV_SEARCH_FILTER to free memory and states associated with the Search operation.

The level of granularity for partial results is an individual attribute value. If the attribute is a multivalued attribute and its values are split across two or more calls to **NWDSSearch**, the current object name, object info, and attribute name is repeated in each subsequent result buffer.

For information about how to conduct a search and for more details about NDS searches, see Searching NDS and NDS Search Introduction.

NOTE: Currently, because of aliasing, searching a subtree can result (1) in duplicate entries or (2) in an infinite loop.

NOTE: On large networks, iterative processes, like **NWDSSearch**, might take a lot of time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. These processes can be interrupted or aborted by calling the **NWDSCloseIteration** function.

Developers should call the **NWDSCloseIteration** function to allow users of their applications to abort an iterative process that is taking too long to complete.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSCloseIteration, **NWDSAddFilterToken**, **NWDSAllocFilter**, **NWDSDelFilterToken**, **NWDSFreeFilter**, **NWDSPutFilter**

NWDSSetContext

Sets the value of an NDS context variable
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdc.h>

NWDSCCODE N_API NWDSSetContext (
   NWDSContextHandle   context,
   nint                key,
   nptr                value);
```

Pascal Syntax

```
#include <nwdsdc.inc>

Function NWDSSetContext
  (context : NWDSContextHandle;
   key : nint;
   value : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

key

(IN) Specifies the context variable to set.

value

(IN) Points to the value for *context*.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

Applications do not have direct access to the NDS context variables. To change the settings of the context variables, applications must call **NWDSSetContext**, using a key to identify for which context variable to set information. See DSI Flags.

The **NWDSSetContext** function can be called using DCK_NAME_CACHE_DEPTH to query or set the depth of the name cache (how many names the cache will remember for the context handle). See Name Caching.

The *context* parameter usually defaults to the preferred tree name. Under NLM, the *context* parameter defaults to the local server tree name.

The key is identified by the *key* parameter. The following keys are defined in NWDSDC.H:

1	DCK_FLAGS	nuint32	Bit definitions
2	DCK_CONFIDENCE	nuint32	Definitions: `0 DCV_LOW_CONF 1 DCV_MED_CONF 2 DCV_HIGH_CONF`
3	DCK_NAME_CONTEXT		Character string array
4	DCK_TRANSPORT_TYPE	nuint32[2]	Not currently in use
5	DCK_REFERRAL_SCOPE	nuint32	Definitions: `0 DCV_ANY_SCOPE 1 DCV_COUNTRY_SCOPE 2 DCV_ORGANIZATION_SCOPE 3 DCV_LOCAL_SCOPE`
8	DCK_LAST_CONNECTION		Returns NWCONN_HANDLE
11	DCK_TREE_NAME		Character string array of at most NW_MAX_TREE_NAME_LEN (includes NULL) ASCII or UNICODE characters

To call **NWDSSetContext** for multiple NDS identities, set the *key* parameter to DCK_TREE_NAME.

The *value* parameter should point to a variable of a type matching the type of the specified variable.

The flags associated with the DCK_FLAGS key are defined as follows:

0x001 L	$00000001	DCV_DEREF_ALIASES
0x002 L	$00000002	DCV_XLATE_STRINGS
0x004 L	$00000004	DCV_TYPELESS_NAMES
0x008 L	$00000008	DCV_ASYNC_MODE
0x010 L	$00000010	DCV_CANONICALIZE_NAMES
0x040 L	$00000040	DCV_DEREF_BASE_CLASS
0x080 L	$00000080	DCV_DISALLOW_REFERRALS

NOTE: Before setting the context flags, first read in the current settings of the flags by calling **NWDSGetContext**. Then use bitwise operations to change the flag(s) you want to change while leaving the settings of the other flags unchanged. Then call **NWDSSetContext** to set the context flags to the desired settings.

If the *key* parameter is DCK_CONFIDENCE, the value pointed to by the *value* parameter can be one of the following:

```
0    DCV_LOW_CONF
1    DCV_MED_CONF
2    DCV_HIGH_CONF
```

If the *key* parameter is DCK_NAME_CONTEXT, the *value* parameter points to a buffer containing the name context. The maximum size of the buffer is ((MAX_DN_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

If the *key* parameter is DCK_REFERRAL_SCOPE, the *value* parameter can be one of the following:

```
0    DCV_ANY_SCOPE
1    DCV_COUNTRY_SCOPE
2    DCV_ORGANIZATION_SCOPE
3    DCV_LOCAL_SCOPE
```

Values stored in a directory context can be read by calling **NWDSGetContext**.

NCP Calls

None

See Also

NWDSCreateContextHandle, **NWDSFreeContext**, **NWDSGetContext**

NWDSSetDefNameContext

Sets the default name context for a specified tree

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSSetDefNameContext (
   NWDSContextHandle   context,
   nuint               nameContextLen
   pnstr8              nameContext);
```

Pascal Syntax

```
#include <nwdsconn.inc>

Function NWDSSetDefNameContext
  (context : NWDSContextHandle;
   nameContextLen : nuint;
   nameContext : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

nameContextLen

(IN) Specifies the length (in bytes) of the *nameContext* buffer.

nameContext

(IN) Points to the name context value to set as default.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSSetDefNameContext sets the default name context for the tree specified in the context (or if the tree name isn't set, the preferred tree name).

NWDSSetDefNameContext differs from **NWSetDefaultNameContext** in that **NWDSSetDefNameContext** has an added parameter, *context*, and operates on a per tree basis. Also, **NWDSSetDefNameContext** can return the name context in Unicode while **NWSetDefaultNameContext** could only return the data in local code page format.

The default name context for the preferred tree can be set by the DEFAULT NAME CONTEXT configuration parameter, or by calling either **NWDSSetDefNameContext** or **NWSetDefaultNameContext**. The default name context for another tree (different from the preferred tree) can be set by calling **NWDSSetDefNameContext**.

The default name context can be from 0 to 257 bytes long for local code page strings (including the NULL), or 0 to 514 bytes long for Unicode strings (including the 2 bytes for NULL). If the *nameContext* buffer is too large, an error is returned and no data is copied.

If the underlying requester does not support multiple NDS trees, the default name context for the default tree will be returned (that is, the tree name specified in the context will be ignored).

NCP Calls

None

See Also

NWGetDefaultNameContext, **NWDSGetDefNameContext**, **NWSetDefaultNameContext**

NWDSSetMonitoredConnection

Tracks the connection
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwdstype.h>
#include <nwndscon.h>

NWCCODE N_API NWDSSetMonitoredConnection
  (NWCONN_HANDLE   conn);
```

Pascal Syntax

```
Function NWDSSetMonitoredConnection
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the connection handle of the desired connection.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
negative value	UNSUCCESSFUL

Remarks

When a user logs into the directory for directory service calls, several attributes are created and maintained on the NetWare server where the user's object resides. If the connection is removed, these attributes are destroyed. To prevent these attributes from being destroyed, call **NWDSSetMonitoredConnection** to track the connection. If the connection is destroyed, several NDS functions such as **NWDSWhoAmI** do not return valid information.

To call **NWDSSetMonitoredConnection** in DOS or Windows 3.1, VLMs must be running. NETX does not support **NWDSSetMonitoredConnection** and returns an error if VLMs are not running.

NCP Calls

None

See Also

NWDSOpenMonitoredConn

NWDSSplitPartition

Divides a partition into two partitions at a specified object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSSplitPartition (
   NWDSContextHandle   context,
   pnstr8              subordinatePartition,
   nflag32             flags);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSSplitPartition
  (context : NWDSContextHandle;
   subordinatePartition : pnstr8;
   flags : nflag32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

subordinatePartition

(IN) Points to the name of the object that will be the root of the subordinate partition to be split.

flags

(IN) Reserved; pass in 0.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E2	TOO_FEW_FRAGMENTS
0x89E3	TOO_MANY_FRAGMENTS
0x89E4	PROTOCOL_VIOLATION
0x89E5	SIZE_LIMIT_EXCEEDED
0x89FD	UNKNOWN_REQUEST
0x89FD	INVALID_PACKET_LENGTH
0x89FE	BAD_PACKET
0x89FF	Failure not related to NDS
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

To split a partition, there must be a single replica (Master) of the partition to be split.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSAddReplica, **NWDSJoinPartitions**, **NWDSSplitPartition**

NWDSSyncPartition

Signals the skulker to schedule an update of a specified partition a specified number of seconds into the future

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdspart.h>

NWDSCCODE N_API NWDSSyncPartition (
   NWDSContextHandle   context,
   pnstr8              server,
   pnstr8              partition,
   nuint32             seconds);
```

Pascal Syntax

```
#include <nwdspart.inc>

Function NWDSSyncPartition
  (context : NWDSContextHandle;
   server : pnstr8;
   partition : pnstr8;
   seconds : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

server

(IN) Points to the server name where the partition resides.

partition

(IN) Points to the name of the partition to update.

seconds

(IN) Specifies the number of seconds to wait until the partition is updated.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

The partition must reside on the specified server.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSSyncReplicaToServer, **NWDSSyncSchema**

NWDSSyncReplicaToServer

Requests a replica to synchronize with a specific server

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsmisc.h>

NWDSCCODE N_API NWDSSyncReplicaToServer (
   NWDSContextHandle    context,
   NWCONN_HANDLE        connHandle,
   pnstr8               partitionRootName,
   pnstr8               destServerName,
   nuint32              actionFlags,
   nuint32              delaySeconds);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWDSSyncReplicaToServer
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   partitionRootName : pnstr8;
   destServerName : pnstr8;
   actionFlags : nuint32;
   delaySeconds : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

serverName

(IN) Specifies the server that contains the replica being synchronized.

partitionRootName

(IN) Specifies the name of the partition whose replica is being synchronized.

destServerName

(IN) Specifies the server to which the replica should synchronize to.

actionFlags

(IN) Specifies the synchronization action to be taken.

(IN) Specifies the synchronization action to be taken.

delaySeconds

(IN) Specifies the number of seconds to delay before beginning the synchronization.

Return Values

0x0000	SUCCESSFUL
nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSSyncReplicaToServer requests that a replica initiate synchronization with the destination server identified by *destServerName*. The *actionFlags* parameter has the following definition:

SF_DO_IMMEDIATE

Perform the action immediately.

SF_TRANSITION

If the replica is in one of the states "New", "Dying", or "Transition On", the request is ignored and SUCCESS is returned.

NWDSSyncReplicaToServer has the side effect of blocking until the synchronization process has completed. The return code indicates the status of the replica by returning SUCCESS or a negative error code, which indicates a problem with synchronization of this replica.

NCP Calls

None

See Also

NWDSSyncPartition

NWDSSyncSchema

Signals the skulker to schedule an update of the schema a specified number of seconds in the future

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdssch.h>

NWDSCCODE N_API NWDSSyncSchema (
   NWDSContextHandle   context,
   pnstr8              server,
   nuint32             seconds);
```

Pascal Syntax

```
#include <nwdssch.inc>

Function NWDSSyncSchema
  (context : NWDSContextHandle;
   server : pnstr8;
   seconds : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

server

(IN) Points to the server name to signal.

seconds

(IN) Specifies the number of seconds to wait until the update is scheduled.

Return Values

0x0000	SUCCESSFUL

negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWDSSyncSchema wakes up the sleeping synchronization process and alerts it to begin synchronization at the time specified.

NCP Calls

0x2222 39 0 Synchronize Schema

See Also

NWDSSyncPartition

NWDSUnlockConnection

Enables the connection to be placed on the LRU list and unlicenses the connection if no other resources are allocated

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWDSUnlockConnection
  (NWCONN_HANDLE   conn);
```

Pascal Syntax

```
Function NWDSUnlockConnection
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the connection handle for the connection to unlock.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL

Remarks

If there are no other tasks having resouces on the connection, and the connection is licensed, the connection will be unlicensed on the NetWare server.

The connection is licensed by calling **NWCCLicenseConn**. For the connection to be licensed, it has to be authenticated.

NCP Calls

None

See Also

NWCCGetConnInfo, **NWCCLicenseConn**, **NWDSAuthenticate**

NWDSVerifyObjectPassword

Verifies the password of an object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsasa.h>

NWDSCCODE N_API NWDSVerifyObjectPassword (
   NWDSContextHandle   contextHandle,
   nflag32             optionsFlag,
   pnstr8              objectName,
   pnstr8              password);
```

Pascal Syntax

```
#include <nwdsasa.inc>

Function NWDSVerifyObjectPassword
  (context : NWDSContextHandle;
   optionsFlag : nflag32;
   objectName : pnstr8;
   password : pnstr8
) : NWDSCCODE;
```

Parameters

contextHandle

(IN) Specifies the handle to the name context structure.

optionsFlag

(IN) Reserved; pass in zero.

objectName

(IN) Points to the object name (under the context) of the object to verify.

password

(IN) Points to the clear-text password for the object.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

To call **NWDSVerifyObjectPassword** successfully, the current password of the object must be known. If no such password exists, *password* should point to a zero-length string. All strings used by **NWDSVerifyObjectPassword** are NULL-terminated. *password* can be any length and all characters are significant. Upper- and lowercase letters are distinct.

The password does not appear on the wire in **NWDSVerifyObjectPassword**. It is used to decrypt the private key attribute of the object. *password* is overwritten by **NWDSVerifyObjectPassword** to prevent compromising it locally. If an application has copied the password, it should destroy (overwrite) any copies as soon as possible.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP
0x2222 104 02	Send NDS Fragmented Request/Reply

See Also

NWDSGenerateObjectKeyPair, **NWDSLogin**, **NWDSChangeObjectPassword**

NWDSWhoAmI

Returns the distinguished name of the object currently logged in to NDS
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NWDSCCODE N_API NWDSWhoAmI (
   NWDSContextHandle   context,
   pnstr8              objectName);
```

Pascal Syntax

```
#include <nwdsdsa.inc>

Function NWDSWhoAmI
  (context : NWDSContextHandle;
   objectName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(OUT) Points to the distinguished name of the object logged in to NDS.

Return Values

0x0000	SUCCESSFUL
negative value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

If the object is not currently logged in, **NWDSWhoAmI** returns an error.

Whether the name returned in *objectName* as a full name or a partial name depends upon the setting of the context flag associated with DCV_CANONICALIZE_NAMES. If the DCV_CANONICALIZE_NAMES flag is set to ON, **NWDSWhoAmI** returns a partial name. If the DCV_CANONICALIZE_NAMES flag is set to OFF, **NWDSWhoAmI** returns a full name.

If the context flag associated with DCV_TYPELESS_NAMES is set to ON, the name returned by **NWDSWhoAmI** will be untyped, otherwise it will be typed.

The caller must allocate memory to hold the distinguished name. The size of memory allocated is (MAX_DN_CHARS+1)*sizeof(character size), where character size is 1 for single-byte characters and 2 for double-byte characters (Unicode is double byte). One character is used for NULL termination.

NCP Calls

None

NWGetDefaultNameContext

Allows the user to get the default name context
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h
see also
#include <nwndscon.h>

NWCCODE N_API NWGetDefaultNameContext (
   nuint16        bufferSize,
   nuint8 N_FAR  *context);
```

Pascal Syntax

```
Function NWGetDefaultNameContext
  (bufferSize : nuint16;
   context : pnuint8
) : NWCCODE;
```

Parameters

bufferSize
(IN) Specifies the maximum size of buffer.

context
(OUT) Points to a buffer retrieving the 256-byte default name context. A NULL-terminated string containing the name context is returned.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8833	INVALID_BUFFER_LENGTH

Remarks

The name may have been set originally in NET.CFG, or it could be set by calling **NWSetDefaultNameContext**. If the name context is empty, a NULL string is returned.

To call **NWGetDefaultNameContext** in DOS or Windows, VLMs must be running. NETX does not support**NWGetDefaultNameContext** and will return an error if VLMs are not running.

NCP Calls

None

See Also

NWSetDefaultNameContext

NWGetFileServerUTCTime

Returns the Coordinated Universal Time (UTC) setting of a server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

nint N_API NWGetFileServerUTCTime (
   NWCONN_HANDLE   conn,
   pnuint32        time);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWGetFileServerUTCTime
  (conn : NWCONN_HANDLE;
   time : pnuint32
) : nint;
```

Parameters

conn

(IN) Specifies the connection handle to the server whose time needs to be retrieved.

time

(OUT) Points to the time setting (in UTC time) of the server.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0xFD6D	ERR_TIME_NOT_SYNCHRONIZED
Negative	Negative values indicate errors. See NDS Values (-001 to

value	-699).

Remarks

NWGetFileServerUTCTime will determine the time setting on remote and local servers. (The **time** function only returns the time setting for local servers.)

The time placed in the location pointed to by the *time* parameter represents the time in seconds since January 1, 1970 (Coordinated Universal Time).

NCP Calls

0x2222 114 1	Get UTC Time

NWGetNextConnectionID (obsolete 6/96)

but is now obsolete. Call **NWCCScanConnRefs** instead.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWGetNextConnectionID
  (NWCONN_HANDLE N_FAR *  conn);
```

Pascal Syntax

```
Function NWGetNextConnectionID
  (Var conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Points to a connection handle of the server specified by server name.

Return Values

0x0000	SUCCESSFUL
0x8801	No More Connections
0x8801	INVALID_CONNECTION

Remarks

On the first call, pass zero (0) in for *conn*. On subsequent calls pass in the last *conn*.

To call **NWGetNextConnectionID (obsolete 6/96)** in DOS or Windows, VLMs must be running. NETX does not support **NWGetNextConnectionID (obsolete 6/96)** and will return an error if VLMs are not running.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP

See Also

NWCCScanConnRefs, **NWCCOpenConnByRef**, **NWCCGetConnInfo**

NWGetNumConnections

Returns the number of connections that can be supported by VLM
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwdstype.h>
#include <nwndscon.h>

NWCCODE N_API NWGetNumConnections (
   nuint16 N_FAR  *numConnections);
```

Pascal Syntax

```
Function NWGetNumConnections
  (numConnections : pnuint16
) : NWCCODE;
```

Parameters

numConnections

(OUT) Points to the number of connections supported.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8800	VLM_ERROR

Remarks

The number of connections can be configured in the net.cfg file.

To call **NWGetNumConnections** in DOS or in Windows, VLMs must be running. NETX does not support **NWGetNumConnections** and will return an error if VLMs are not running.

NCP Calls

None

NWGetNWNetVersion

Returns the NWNet library version number

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsmisc.h>

N_EXTERN_LIBRARY(void) NWGetNWNetVersion (
   nuint8 N_FAR  *majorVersion,
   nuint8 N_FAR  *minorVersion,
   nuint8 N_FAR  *revisionLevel,
   nuint8 N_FAR  *betaReleaseLevel);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWGetNWNetVersion
  (majorVersion : pnuint8;
   minorVersion : pnuint8;
   revisionLevel : pnuint8;
   betaReleaseLevel : pnuint8
);
```

Parameters

majorVersion

(OUT) Points to the major version number.

minorVersion

(OUT) Points to the minor version number.

revisionLevel

(OUT) Points to the revision level number.

betaReleaseLevel

(OUT) Points to the beta release level number.

NCP Calls

None

See Also

NWDSGetDSVerInfo

NWGetPreferredConnName

Gets the name of the preferred connection
NetWare Server: N/A
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWGetPreferredConnName (
   nuint8 N_FAR  *preferredName,
   nuint8 N_FAR  *preferredType);
```

Pascal Syntax

```
Function NWGetPreferredConnName
   (preferredName : pnuint8;
    preferredType : pnuint8
) : NWCCODE;
```

Parameters

preferredName

(OUT) Points to the buffer where the preferred name is stored.

preferredType

(OUT) Points to the preferred name type set [NWNDS_CONNECTION = 1 (Preferred Tree Name) or 0 (Preferred Server)].

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL

Remarks

NWGetPreferredConnName will only work if VLMs are loaded; it will not work with NETX.

If both *preferredType* names are set by API calls or NET.CFG, the order is determined at VLM load time.

Defaults are Preferred Tree Name, then Preferred Server.

If a Preferred Tree Name is not specified, the Preferred Server will be returned and *preferredType* will be zero (Preferred Server). However, if a Preferred Tree Name is specified in NET.CFG, or if **NWSetPreferredDSTree** is called, *preferredName* will be the Preferred Tree Name and the server type will be set to NWNDS_CONNECTION = 1 (Preferred Tree Name). If BIND.VLM is loaded before NDS.VLM, the opposite is true.

NCP Calls

None

See Also

NWSetPreferredDSTree, **NWGetPreferredDSServer (obsolete 6/96)**

NWGetPreferredDSServer (obsolete 6/96)

but is now obsolete. Call **NWGetPreferredConnName** followed by calling **NWCCOpenConnByName** instead.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet>
see also
#include <nwdstype.h>
#include <nwndscon.h>

NWCCODE N_API NWGetPreferredDSServer
  (NWCONN_HANDLE N_FAR *   conn);
```

Pascal Syntax

```
Function NWGetPreferredDSServer
  (Var conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(OUT) Points to the NDS connection handle.

Return Values

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8855	PREFERRED_NOT_FOUND
0x8859	NO_PREFERRED_SPECIFIED

Remarks

Remarks

NWGetPreferredDSServer (obsolete 6/96) returns a connection only if a preferred tree has been set in either NET.CFG or by calling **NWSetPreferredDSTree**. If a preferred tree is not set, 0x8859 is returned.

In OS/2, **NWGetPreferredDSServer (obsolete 6/96)** returns a server connection ID in the tree of the specified preferred tree, but not specifically the preferred server.

In DOS or Windows,**NWGetPreferredDSServer (obsolete 6/96)** returns the first NDS connection ID in the connection table, if there is one; otherwise it attaches to a server in the specified tree.

To call **NWGetPreferredDSServer (obsolete 6/96)** in DOS or in Windows, VLMs must be running. NETX does not support **NWGetPreferredDSServer (obsolete 6/96)** and will return an error if VLMs are not running.

NOTE: **NWGetPreferredDSServer (obsolete 6/96)** returns the preferred server's connection handle, which is the case for new requesters like Windows NT and Windows95. Some older requesters like DOS and Windows (VLM requesters) contained a bug, which returned the first connection in the connection table instead. Developers accustomed to the operation of this call using a VLM requester will find that the newer requesters will not return what they have come to expect.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 22	Get Station's Logged Info (old)
0x2222 23 28	Get Station's Logged Info
0x2222 104 01	Ping for NDS NCP

See Also

NWCCLicenseConn, **NWCCOpenConnByAddr**, **NWSetPreferredDSTree**

NWIsDSAuthenticated

Returns whether NDS has credentials for a background authentication in the current NDS tree

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWIsDSAuthenticated (
   void);
```

Pascal Syntax

```
Function NWIsDSAuthenticated (
   Par1 : nptr
) : NWCCODE;
```

Return Values

These are common return values; see Return Values for more information.

0x000 1	Authenticated through NDS
0x000 0	Not authenticated through NDS

Remarks

To call **NWIsDSAuthenticated** in DOS or in Windows, VLMs must be running. NETX does not support **NWIsDSAuthenticated** and will return an error if VLMs are not running.

NCP Calls

None

See Also

NWCCGetConnInfo, NWCCScanConnRefs

NWIsDSServer

Checks presence or absence of NDS on the server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwdsmisc.h>

NWDSCCODE NWFAR NWPASCAL NWIsDSServer (
   NWCONN_HANDLE    conn,
   pnstr8           treeName);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWIsDSServer
  (conn : NWCONN_HANDLE;
   treeName : pnstr8
) : NWDSCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

treeName

(OUT) Points to the tree name returned if NDS.

Return Values

These are common return values; see Return Values for more information.

0x0001	NDS NCP is hooked and NDS is running
0x0000	Not NDS

NCP Calls

0x2222 104 01   Ping for NDS NCP

NWNetInit

Does the initial setup that is necessary before calling any other NDS functions

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwdsmisc.h>

NWDSCCODE N_API NWNetInit (
   nptr   in,
   nptr   out);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWNetInit
  (in : nptr;
   out : nptr
) : NWDSCCODE;
```

Parameters

in

(IN) Points to the input parameter value.

out

(OUT) Points to the output parameter value.

Return Values

0x0000	SUCCESSFUL
nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWNetInit initializes the NDS library. Both of the parameters, *in* and *out*, should be NULL when **NWNetInit** is called.

NCP Calls

None

See Also

NWNetTerm

NWNetTerm

Shuts down and cleans up after NDS
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWNetTerm (
   nptr   reserved);
```

Pascal Syntax

```
#include <nwdsmisc.inc>

Function NWNetTerm (
   reserved : nptr
) : NWDSCCODE;
```

Return Values

0x0000	SUCCESSFUL
nonzero value	Negative values indicate errors. See NDS Values (-001 to -699).

Remarks

NWNetTerm terminates the NDS library.

Under VLM, **NWNetTerm** has no effect and other NDS functions may be called after calling **NWNetTerm**.

Under NLM, OS/2, Windows, Windows95, and Windows NT, **NWNetTerm** should be called last as it will shut down and clean up after NDS.

If, after calling **NWNetTerm**, you wish to call other NDS functions, call **NWNetInit** before calling any other NDS functions.

NCP Calls

None

See Also

NWNetInit

NWSetDefaultNameContext

Sets the default name context

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWSetDefaultNameContext (
   nuint16        contextLength,
   nuint8 N_FAR  *context);
```

Pascal Syntax

```
Function NWSetDefaultNameContext
  (contextLength : nuint16;
   context : pnuint8
) : NWCCODE;
```

Parameters

contextLength

(IN) Specifies the length of the context.

context

(IN) Points to the buffer containing the 256-byte default name context.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8833	INVALID_BUFFER_LENGTH

Remarks

The default name context may have been originally set in NET.CFG. If the name is longer than 256 bytes, it will be truncated.

To call **NWSetDefaultNameContext** in DOS or in Windows, VLMs must be running. NETX does not support**NWSetDefaultNameContext** and will return an error if VLMs are not running.

NCP Calls

None

See Also

NWGetDefaultNameContext

NWSetPreferredDSTree

Sets the preferred NDS Server tree name in the requester's tables
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: NDS

Syntax

```
#include <nwnet.h>
see also
#include <nwdstype.h>
#include <nwndscon.h>

NWCCODE N_API NWSetPreferredDSTree (
   nuint16   length,
   pnuint8   tree);
```

Pascal Syntax

```
Function NWSetPreferredDSTree
  (length : nuint16;
   treeName : pnuint8
) : NWCCODE;
```

Parameters

length
(IN) Specifies the length of the tree name.

tree
(IN) Points to the NDS server tree name.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8836	INVALID_PARAMETER

Remarks

NWSetPreferredDSTree sets a tree name for future NDS functions. The tree name may also be set in NET.CFG. The maximum name length is 32 characters. If the server name is too long, INVALID_PARAMETER is returned.

NWGetPreferredDSServer creates the connection attachment to the server specified in **NWSetPreferredDSTree**.

To call **NWSetPreferredDSTree** in DOS or in Windows, VLMs must be running. NETX does not support **NWSetPreferredDSTree** and will return an error if VLMs are not running.

NCP Calls

None

See Also

NWGetPreferredDSServer (obsolete 6/96)

NDS: Structures

Asn1ID_T

Holds the ASN1 ID of an object
Service: NDS
Defined In: nwdsbuft.h

Structure

```
typedef struct
{
   nuint32   length;
   nuint8    data[MAX_ASN1_NAME];
} Asn1ID_T;
```

Pascal Structure

```
Defined in nwdsbuft.inc

pAsn1ID_T = ^Asn1ID_T;
   Asn1ID_T = Record
    length : nuint32;
    data : Array[0..MAX_ASN1_NAME-1] Of nuint8
  End;
```

Fields

length

data

Attr_Info_T

Contains information used to define attribute types

Service: NDS

Defined In: nwdsbuft.h

Structure

```
typedef struct
{
   nuint32    attrFlags;
   nuint32    attrSyntaxID;
   nuint32    attrLower;
   nuint32    attrUpper;
   Asn1ID_T   asn1ID;
} Attr_Info_T;
```

Pascal Structure

```
Defined in nwdsbuft.inc

pAttr_Info_T = ^Attr_Info_T;
  Attr_Info_T = Record
    attrFlags : nuint32;
    attrSyntaxID : nuint32;
    attrLower : nuint32;
    attrUpper : nuint32;
    asn1ID : Asn1ID_T
  End;
```

Fields

attrFlags

Indicates the constraints assigned to the attribute:

0x 01	$00 01	DS_SINGLE_VALUED_ATTR
0x 02	$00 02	DS_SIZED_ATTR
0x 04	$00 04	DS_NONREMOVABLE_ATTR
0x 08	$00 08	DS_READ_ONLY_ATTR
0x 10	$00 10	DS_HIDDEN_ATTR

0x 20	$00 20	DS_STRING_ATTR
0x 40	$00 40	DS_SYNC_IMMEDIATE

attrSyntaxID

Indicates the syntax ID of the attribute type.

attrLower

Indicates the lower limit of the attribute.

attrUpper

Indicates the upper limit of the attribute.

asn1ID

Indicates the object identifier allocated according to the rules specified in the ASN.1 standard; if no object identifier has been registered for the class, a zero-length octet string is specified.

Back_Link_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   nuint32   remoteID;
   pnstr8    objectName;
} Back_Link_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

  pBack_Link_T = ^Back_Link_T;
   Back_Link_T = Record
      remoteID : nuint32;
      objectName : pnstr8
   End;
```

Fields

remoteID

objectName

Bit_String_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   nuint32   numOfBits;
   nptr      data;
} Bit_String_T;
```

Defined In

```
Defined in nwdsattr.inc

pBit_String_T = ^Bit_String_T;
   Bit_String_T = Record
      numOfBits : nuint32;
      data : pnuint8
   End;
```

Fields

numOfBits

data

Buf_T

Initializes and handles input and output buffers

Service: NDS

Defined In: nwdsbuft.h

Structure

```
typedef struct
{
   nuint32   operation;
   nuint32   flags;
   nuint32   maxLen;
   nuint32   curLen;
   pnuint8   lastCount;
   pnuint8   curPos;
   pnuint8   data;
} Buf_T;
```

Pascal Structure

```
Defined in nwdsbuft.inc

pBuf_T = ^Buf_T;
ppBuf_T = ^pBuf_T;
  Buf_T = Record
    operation : nuint32;
    flags : nuint32;
    maxLen : nuint32;
    curLen : nuint32;
    lastCount : pnuint8;
    curPos : pnuint8;
    data : pnuint8
  End;
```

Fields

operation

Indicates the verb of the function operating on the buffer; set by **NWDSInitBuf**. The operation determines the type of data in the buffer.

flags

Indicates the set of bit flags. Only the first bit is defined. If it is set, the buffer contains input data. If this bit is clear, the buffer contains results:

0x0001 ($00000001) INPUT_BUFFER

maxLen

Indicates the amount of memory allocated when **NWDSAllocBuf** is called. This is the maximum length of data the buffer can contain. This member is set to 0 when the buffer is allocated.

curLen

Indicates the length of the current buffer; manipulated internally by get and put routines only. This member is set to 0 when the buffer is allocated or initialized. For an output buffer, this member is the total bytes of data received by the client.

lastCount

Indicates the number of items in the data currently stored in the buffer; manipulated internally for iterative operations on the buffer.

curPos

Specifies the offset in the data area where the next operation should occur. This member is set to the start of the buffer to which the *data* field points when the buffer is allocated or initialized, or when a result is returned. The "put" and "get" functions update the member. It is manipulated internally.

data

Points to the actual data stored in the buffer.

CI_List_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   struct_ci_list N_FAR *   next;
   pnstr8                   s;
} CI_List_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pCI_List_T = ^CI_List_T;
   CI_List_T = Record
      next : pCI_List_T;
      s : pnstr8
   End;
```

Fields

next

Points to the next node containing a case-ignore string.

s

Indicates a case-ignore string for this node.

Class_Info_T

Contains information used to maintain the schema

Service: NDS

Defined In: nwdsbuft.h

Structure

```
typedef struct
{
   nuint32   classFlags;
   Asn1ID_T   asn1ID;
} Class_Info_T;
```

Pascal Structure

```
Defined in nwdsbuft.inc

pClass_Info_T = ^Class_Info_T;
  Class_Info_T = Record
    classFlags : nuint32;
    asn1ID : Asn1ID_T
  End;
```

Fields

classFlags

0x 01	$0 1	DS_CONTAINER_CLASS
0x 02	$0 2	DS_EFFECTIVE_CLASS
0x 04	$0 4	DS_NONREMOVABLE_CLASS
0x 08	$0 8	DS_AMBIGUOUS_NAMING
0x 10	$1 0	DS_AMBIGUOUS_CONTAINMENT

asn1ID

EMail_Address_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   nuint32   type;
   nptr      address;
} EMail_Address_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

 pEMail_Address_T = ^EMail_Address_T;
   EMail_Address_T = Record
      mailType : nuint32;
      address : pnstr8
   End;
```

Fields

type

address

EMAIL_ADDRESS_TYPE

Service: NDS
Defined In: nwdsdefs.h

Structure

```
typedef enum
{
   SMF70 = 1,
   SMF71
} EMAIL_ADDRESS_TYPE
```

Fields

Fax_Number_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   pnstr8         telephoneNumber;
   Bit_String_T   parameters;
} Fax_Number_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pFax_Number_T = ^Fax_Number_T;
   Fax_Number_T = Record
      telephoneNumber : pnstr8;
      parameters : Bit_String_T
   End;
```

Fields

telephoneNumber

Points to the next node containing a case-ignore string.

parameters

Indicates a case-ignore string for this node.

Filter_Cursor_T

Builds an expression tree to search for objects in NDS
Service: NDS
Defined In: nwdsfilt.h

Structure

```
typedef struct
{
   pFilter_Node_T   fn;
   nuint16          level;
   nuint32          expect;
} Filter_Cursor_T;
```

Defined In

```
Defined in nwdsfilt.inc

pFilter_Cursor_T = ^Filter_Cursor_T;
ppFilter_Cursor_T = ^pFilter_Cursor_T;
   Filter_Cursor_T = Record
      fn : pFilter_Node_T;
      level : nuint16;
      expect : nuint32
   End;
```

Fields

fn

Indicates the address of the current node structure in the expression tree.

level

Indicates the number of nodes superior to the current node plus 1.

expect

Indicates which tokens are legitimate values for the current node with a bit-map.

Filter_Node_T

Builds an expression tree to search for objects in NDS
Service: NDS
Defined In: nwdsfilt.h

Structure

```
typedef struct
{
   struct_filter_node N_FAR *   parent;
   struct_filter_node N_FAR *   left;
   struct_filter_node N_FAR *   right;
   nptr                         value;
   nuint32                      syntax;
   nuint16                      token;
} Filter_Node_T;
```

Pascal Structure

```
Defined in nwdsfilt.inc

pFilter_Node_T = ^Filter_Node_T;
   Filter_Node_T = Record
    parent : pFilter_Node_T;
    left : pFilter_Node_T;
    right : pFilter_Node_T;
    value : nptr;
    syntax : nuint32;
    token : nuint16
  End;
```

Fields

parent

Indicates the address of the parent node. Refers to nodes in relation to the currently selected node.

left

Indicates the address of the left subordinate. Refers to nodes in relation to the currently selected node.

right

Indicates the address of the right subordinate. Refers to nodes in relation to the currently selected node.

value

Indicates the address of an attribute name of attribute value, if*token* is a value or a name.

a value or a name.

syntax

Indicates the syntax associated with the value of *token*.

token

Indicates the type of node. If *token* specifies neither an attribute name (14) nor an attribute value (6), *value* and *syntax* members are ignored:

```
0 FTOK_END
1 FTOK_OR
2 FTOK_AND
3 FTOK_NOT
4 FTOK_LPAREN
5 FTOK_RPAREN
6 FTOK_AVAL
7 FTOK_EQ
8 FTOK_GE
9 FTOK_L
10 FTOK_APPROX
14 FTOK_ANAME
15 FTOK_PRESENT
16 FTOK_RDN
17 FTOK_BASECLS
```

Hold_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   pnstr8    objectName;
   nuint32   amount;
} Hold_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pHold_T = ^Hold_T;
   Hold_T = Record
      objectName : pnstr8;
      amount : nuint32
   End;
```

Fields

objectName

amount

NAME_SPACE_TYPE

Service: NDS
Defined In: nwdsdefs.h

Structure

```
typedef enum
{
   DS_DOS,
   DS_MACINTOSH,
   DS_UNIX,
   DS_FTAM,
   DS_OS2
} NAME_SPACE_TYPE;
```

Pascal Structure

```
Defined in nwdsdefs.inc

  NAME_SPACE_TYPE = (
      DS_DOS, DS_MACINTOSH, DS_UNIX, DS_FTAM, DS_OS2
 );
```

Fields

NDSStatsInfo_T

Contains statistical information for NDS relative to an NDS server

Service: NDS

Defined In: nwdsmisc.h

Structure

```
typedef struct
{
   nuint32   statsVersion;
   nuint32   noSuchEntry;
   nuint32   localEntry;
   nuint32   typeReferral;
   nuint32   aliasReferral;
   nuint32   resetTime;
   nuint32   transportReferral;
   nuint32   upReferral;
   nuint32   downReferral;
   nuint32   requestCount;
   nuint32   requestDataSize;
   nuint32   replyDataSize;
} NDSStatsInfo_T, N_FAR *pNDSStatsInfo_T;
```

Pascal Structure

```
Defined in nwdsmisc.inc

  NDSStatsInfo_T = Record
      statsVersion : nuint32;
      noSuchEntry : nuint32;
      localEntry : nuint32;
      typeReferral : nuint32;
      aliasReferral : nuint32;
      requestCount : nuint32;
      requestDataSize : nuint32;
      replyDataSize : nuint32;
      resetTime : nuint32;
      transportReferral : nuint32;
      upReferral : nuint32;
      downReferral : nuint32
   End;
```

Fields

statsVersion

Indicates the supported members of the statistics structure as it is expanded.

noSuchEntry

Indicates the number of times name ressolution resulted in not locating the entry local to this server.

localEntry

Indicates the number of times name resolution resulted in finding the entry local to this server.

typeReferral

Indicates the number of times name resolution found a local entry, but another replica type was requested.

aliasReferral

Indicates the number of times name resolution responded with an alias referral.

resetTime

Records the last time NDS statistics were reset. The value consists of a whole number of seconds since 12:00 midnight, January 1, 1970, UTC.

transportReferral

Indicates the number of times name resolution located a local entry, but the referral specified does not have an acceptable transport type.

upReferral

Indicates the number of times name resolution was not walking the tree for the caller, and the referral went "up" the tree.

downReferral

Indicates the number of times name resolution was not walking the tree for the caller, and the referral went "down" the tree.

requestCount

Indicates the number of NDS requests received from a remote client (including the NDS client agent used for skulking, etc.).

requestDataSize

Records the sum of request buffer sizes. This number is likely to wrap (overflow back to a lower number) over time.

replayDataSize

Records the sum of reply buffer sizes. This number is likely to wrap (overflow back to a lower number) over time.sport type.

Net_Address_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   nuint32   addressType;
   nuint32   addressLength;
   nptr      address;
} Net_Address_T;
```

Defined In

```
Defined in nwdsattr.inc

pNet_Address_T = ^Net_Address_T;
   Net_Address_T = Record
      addressType : nuint32;
      addressLength : nuint32;
      address : pnuint8
   End;
```

Fields

addressType

Indicates the type of communications protocol used, such as IPX or AppleTalk.

addressLength

Indicates the address length expressed in bytes.

address

Indicates the hexadecimal address. *address* is stored as a binary string; each 4-bit nibble must be converted to hexadecimal before it can be displayed as a hexadecimal address.

NET_ADDRESS_TYPE

Service: NDS
Defined In: nwdsdefs.h

Structure

```
typedef enum
{
   NT_IPX,
   NT_IP,
   NT_SDLC,
   NT_TOKENRING_ETHERNET,
   NT_OSI,
   NT_APPLETALK,
   NT_COUNT
} NET_ADDRESS_TYPE;
```

Pascal Structure

```
Defined in nwdsdefs.inc

   NET_ADDRESS_TYPE = (
      NT_IPX, NT_IP, NT_SDLC, NT_TOKENRING_ETHERNET, NT_OSI, NT_APPLETA
      NT_COUNT
   );
```

Fields

NWDS_TimeStamp_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   nuint32   wholeSeconds;
   nuint16   replicaNum;
   nuint16   eventID;
} NWDS_TimeStamp_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pNWDS_TimeStamp_T = ^NWDS_TimeStamp_T;
   NWDS_TimeStamp_T = Record
      wholeSeconds : nuint32;
      eventID : nuint32
   End;
```

Fields

wholeSeconds

Indicates the value of the time stamp in whole seconds. Zero equals 12:00 a.m. (midnight), January 1, 1970 GMT.

wholeSeconds

eventID

Indicates the order events occur within the whole second interval by an integer value.

Object_ACL_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   pnstr8    protectedAttrName;
   pnstr8    subjectName;
   nuint32   privileges;
} Object_ACL_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pObject_ACL_T = ^Object_ACL_T;
   Object_ACL_T = Record
      protectedAttrName : pnstr8;
      subjectName : pnstr8;
      privileges : nuint32
   End;
```

Fields

protectedAttrName

Indicates the name of the specific attribute to be protected; otherwise, should be null to protect the entire object:

```
"[All Attributes Rights]"   DS_ALL_ATTRS_NAME
"[SMS Rights]"              DS_SMS_RIGHTS_NAME
"[Entry Rights]"            DS_ENTRY_RIGHTS_NAME
```

subjectName

Indicates the name of the object receiving the rights to the protected object:

```
"[Root]"                DS_ROOT_NAME
"[Public]"              DS_PUBLIC_NAME
"[Inheritance Mask]"    DS_MASK_NAME
"[Creator]"             DS_CREATOR_NAME
"[Self]"                DS_SELF_NAME
```

DS_CREATOR_NAME and DS_SELF_NAME can be used only with **NWDSAddObject**.

privileges

Indicates a bit mask identifying specific rights.

Object_Info_T

Handles information used to maintain objects

Service: NDS

Defined In: nwdsbuft.h

Structure

```
typedef struct
{
   nuint32   objectFlags;
   nuint32   subordinateCount;
   time_t    modificationTime;
   char      baseClass[MAX_SCHEMA_NAME_BYTES+2];
} Object_Info_T;
```

Pascal Structure

```
Defined in nwdsbuft.inc

Object_Info_T = Record
   objectFlags : nuint32;
   subordinateCount : nuint32;
   modificationTime : time_t;
   baseClass : Array [0..MAX_SCHEMA_NAME_BYTES+1] Of char;
End;
```

Fields

objectFlags

Indicates the object flags.

subordinateCount

Indicates the number of objects subordinates to the object.

modificationTime

Indicates the time when the object was last modified.

baseClass

Indicates the class used to create the object.

Remarks

The *objectFlags* field can have the following values:

0x000 1	$0001	DS_ALIAS_ENTRY

0x0002	$0002	DS_PARTITION_ROOT
0x0004	$0004	DS_CONTAINER_ENTRY
0x0008	$0008	DS_CONTAINER_ALIAS
0x0010	$0010	DS_MATCHES_LIST_FILTER
0x0020	$0020	DS_REFERENCE_ENTRY
0x0040	$0040	DS_40X_REFERENCE_ENTRY
0x0080	$0080	DS_BACKLINKED
0x0100	$0100	DS_NEW_ENTRY

Octet_String_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   nuint32   length;
   nptr      data;
} Octet_String_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pOctet_String_T = ^Octet_String_T;
   Octet_String_T = Record
      length : nuint32;
      data : pnuint8
   End;
```

Fields

length

data

Path_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   nuint32   nameSpaceType;
   pnstr8    volumeName;
   pnstr8    path;
} Path_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pPath_T = ^Path_T;
   Path_T = Record
      nameSpaceType : nuint32;
      volumeName : pnstr8;
      path : pnstr8
   End;
```

Fields

nameSpaceType

volumeName

path

Replica_Pointer_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   pnstr8          serverName;
   nint32          replicaType;
   nint32          replicaNumber;
   nuint32         count;
   Net_Address_T   replicaAddressHint[1];
} Replica_Pointer_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pReplica_Pointer_T = ^Replica_Pointer_T;
   Replica_Pointer_T = Record
      serverName : pnstr8;
      replicaType : nint32;
      replicaNumber : nint32;
      count : nuint32;
      replicaAddressHint : Array[0..0] Of Net_Address_T
   End;
```

Fields

serverName

Indicates the complete name of the NetWare server storing the replica.

replicaType

Indicates the capabilities of this copy of the partition (Master, Secondary, Read-Only).

replicaNumber

Indicates the number of the replica.

count

Indicates the number of existing replicas.

replicaAddressHint

Indicates the node at which the NetWare Server probably exists.

REPLICA_TYPE

Identifies the values for replica types

Service: NDS

Defined In: nwdsdefs.h

Structure

```
typedef enum
{
   RT_MASTER,
   RT_SECONDARY,
   RT_READONLY,
   RT_SUBREF,
   RT_COUNT
} REPLICA_TYPE;
```

Pascal Structure

```
Defined in nwdsdefs.inc

   REPLICA_TYPE = (
      RT_MASTER, RT_SECONDARY, RT_READONLY, RT_SUBREF, RT_COUNT
   );
```

Fields

SYNTAX

Service: NDS
Defined In: nwdsdefs.h

Structure

```
typedef enum
{
   SYN_UNKNOWN,                  /* 0  */
   SYN_DIST_NAME,                /* 1  */
   SYN_CE_STRING,                /* 2  */
   SYN_CI_STRING,                /* 3  */
   SYN_PR_STRING,                /* 4  */
   SYN_NU_STRING,                /* 5  */
   SYN_CI_LIST,                  /* 6  */
   SYN_BOOLEAN,                  /* 7  */
   SYN_INTEGER,                  /* 8  */
   SYN_OCTET_STRING,             /* 9  */
   SYN_TEL_NUMBER,               /* 10 */
   SYN_FAX_NUMBER,               /* 11 */
   SYN_NET_ADDRESS,              /* 12 */
   SYN_OCTET_LIST,               /* 13 */
   SYN_EMAIL_ADDRESS,            /* 14 */
   SYN_PATH,                     /* 15 */
   SYN_REPLICA_POINTER,          /* 16 */
   SYN_OBJECT_ACL,               /* 17 */
   SYN_PO_ADDRESS,               /* 18 */
   SYN_TIMESTAMP,                /* 19 */
   SYN_CLASS_NAME,               /* 20 */
   SYN_STREAM,                   /* 21 */
   SYN_COUNTER,                  /* 22 */
   SYN_BACK_LINK,                /* 23 */
   SYN_TIME,                     /* 24 */
   SYN_TYPED_NAME,               /* 25 */
   SYN_HOLD,                     /* 26 */
   SYN_INTERVAL,                 /* 27 */
   SYNTAX_COUNT                  /* 28 */
} SYNTAX;
```

Pascal Structure

```
Defined in nwdsdefs.inc

 SYNTAX = (
      SYN_UNKNOWN,                 (* 0 *)
      SYN_DIST_NAME,               (* 1 *)
      SYN_CE_STRING,               (* 2 *)
      SYN_CI_STRING,               (* 3 *)
      SYN_PR_STRING,               (* 4 *)
      SYN_NU_STRING,               (* 5 *)
      SYN_CI_LIST,                 (* 6 *)
      SYN_BOOLEAN,                 (* 7 *)
      SYN_INTEGER,                 (* 8 *)
      SYN_OCTET_STRING,            (* 9 *)
      SYN_TEL_NUMBER,              (* 10 *)
      SYN_FAX_NUMBER,              (* 11 *)
      SYN_NET_ADDRESS,             (* 12 *)
      SYN_OCTET_LIST,              (* 13 *)
      SYN_EMAIL_ADDRESS,           (* 14 *)
      SYN_PATH,                    (* 15 *)
      SYN_REPLICA_POINTER,         (* 16 *)
      SYN_OBJECT_ACL,              (* 17 *)
      SYN_PO_ADDRESS,              (* 18 *)
      SYN_TIMESTAMP,               (* 19 *)
      SYN_CLASS_NAME,              (* 20 *)
      SYN_STREAM,                  (* 21 *)
      SYN_COUNTER,                 (* 22 *)
      SYN_BACK_LINK,               (* 23 *)
      SYN_TIME,                    (* 24 *)
      SYN_TYPED_NAME,              (* 25 *)
      SYN_HOLD,                    (* 26 *)
      SYN_INTERVAL,                (* 27 *)
      SYNTAX_COUNT                 (* 28 *)
   );
```

Fields

Syntax_Info_T

Contains syntax information

Service: NDS

Defined In: nwdsbuft.h

Structure

```
typedef struct
{
   nuint32   ID;
   nstr8     defStr[MAX_SCHEMA_NAME_BYTES + 2];
   nflag16   flags;
} Syntax_Info_T;
```

Pascal Structure

```
Defined in nwdsbuft.inc

pSyntax_Info_T = ^Syntax_Info_T;
  Syntax_Info_T = Record
    ID : nuint32;
    defStr : Array[0..MAX_SCHEMA_NAME_BYTES+2] Of char;
    flags : nflag16
  End;
```

Fields

ID

Indicates the byte representation of the syntax name.

defStr

Indicates the byte representation of the syntax name.

flags

Indicates the matching rules for the syntax such as equality, greater than, less than, etc.

TimeStamp_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   nuint32   wholeSeconds;
   nuint16   replicaNum;
   nuint16   eventID;
} TimeStamp_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pTimeStamp_T = ^TimeStamp_T;
   TimeStamp_T = Record
      wholeSeconds : nuint32;
      replicaNum : nuint16;
      eventID : nuint16
   End;
```

Fields

wholeSeconds

Indicates the whole number of seconds, where zero equals 12:00, midnight, January 1, 1970, UTC.

replicaNum

Indicates the number of the replica on which the event occurred.

eventID

Indicates an integer further ordering events occurring within the same whole-second interval.

Remarks

Two time stamps values are compared by using *wholeSeconds* first and *eventID* second. If *wholeSeconds* are unequal, the order is determined by *wholeSeconds* alone. If *wholeSeconds* are equal and *eventID* are unequal, the order is determined by *eventID*. If *wholeSeconds* and *eventID* are both equal, the time stamps are equal.

When filling out TimeStamp_T, set *eventID* to zero, *replicaNum* to zero, and *wholeSeconds* to the appropriate value.

Typed_Name_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   pnstr8    objectName;
   nuint32   level;
   nuint32   interval;
} Typed_Name_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pTyped_Name_T = ^Typed_Name_T;
   Typed_Name_T = Record
      objectName : pnstr8;
      level : nuint32;
      interval : nuint32
   End;
```

Fields

objectName

level

Indicates the priority of the attribute. This is a relative value assigned by the user.

interval

Indicates the frequency of reference. This is a relative value assigned by the user.

Unknown_Attr_T

Service: NDS
Defined In: nwdsattr.h

Structure

```
typedef struct
{
   pnstr8    attrName;
   nuint32   syntaxID;
   nuint32   valueLen;
   nptr      value;
} Unknown_Attr_T;
```

Pascal Structure

```
Defined in nwdsattr.inc

pUnknown_Attr_T = ^Unknown_Attr_T;
   Unknown_Attr_T = Record
      attrName : pnstr8;
      syntaxID : nuint32;
      valueLen : nuint32;
      value : nptr
   End;
```

Fields

attrName

syntaxID

valueLen

value

NDS Schema

NDS Object Class Definitions

AFP Server

Identifies objects that are classified to provide AFP services.
NetWare Versions: 4.x
Type: Effective

Super Classes

*Top
Server (Class)

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
*Object Class

Optional Attributes

*Account Balance
*ACL
*Allow Unlimited Credit
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
*Equivalent To Me (4.1)
*Full Name
*Host Device
*L (Locality)
*Last Referenced Time (4.1)
*Minimum Account Balance

*Network Address
*O (Organization)
*Obituary
*OU (Organizational Unit)
*Private Key
*Public Key
*Reference
*Resource (Attribute)
*Revision (4.01)
*Security Equals (4.1)
*Security Flags (4.1)
*See Also
Serial Number
*Status
Supported Connections
*User (Attribute)
*Version

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
*[Public]	Read	Network Address
*[Self]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Alias

Defines alias objects. The class of the aliased object determines how the alias is named and where it may be contained.

NetWare Versions: 4.x

Type: Effective

Super Classes

Top

Containment

(special)

Named By

(special)

Mandatory Attributes

Aliased Object Name
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Equivalent To Me (4.1)
*Last Referenced Time (4.1)
*Obituary
*Reference
*Revision (4.01)

Default ACL Template

Object Name	Default Rights	Affected Attributes

*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

An alias is a name containing at least one RDN that is an alias entry. Aliases permit Directory entries to have multiple immediate superiors and consequently provide a basis for alternative names.

Just as the distinguished name of an object expresses the object's relationship to some hierarchy of objects, an alias typically expresses an alternative relationship to a different hierarchy of objects.

An object in the Directory tree may have zero or more aliases. It follows that several alias entries may point to the same object entry. Only object entries can have aliases. Aliases of aliases are not permitted.

An object entry does not have to be a leaf entry to have an alias. Alias entries, however, cannot have subordinates and are always leaf entries.

The Directory uses the *Aliased Object Name* attribute in an alias entry to identify and to find the corresponding object entry.

The object class Alias does not specify naming attributes for alias entries, nor does the class define where in the Directory tree alias entries may be contained. The Directory enforces the naming and containment rules mandated by the base class of the object to which the alias points.

Bindery Object

Used to represent an object that has been created by the Bindery Emulator to emulate a Bindery object.

NetWare Versions: 4.x

Type: Effective

Super Classes

Top

Containment

Organization
Organizational Unit

Named By

Bindery Type
CN (Common Name)

Mandatory Attributes

Bindery Object Restriction
Bindery Type
CN (Common Name)
*Object Class

Optional Attributes

(Special)
*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Equivalent To Me (4.1)
*Last Referenced Time (4.1)
*Obituary
*Reference
*Revision (4.01)

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Bindery Queue

Represents an object that has been created by the Bindery Emulator to emulate a user-defined queue object.

NetWare Versions: 4.x

Type: Effective

Super Classes

*Top
*Resource (Class)
Queue (Class)

Containment

*Organization
*Organizational Unit

Named By

Bindery Type
CN (Common Name)

Mandatory Object Class

Bindery Type
CN (Common Name)
*Object Class
*Queue Directory

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
*Device (Attribute)
*Equivalent To Me (4.1)
*Host Resource Name

*Host Server
*L (Locality)
*Last Referenced Time (4.1)
*Network Address
*O (Organization)
*Obituary
*Operator
*OU (Organizational Unit)
*Reference
*Revision (4.01)
*See Also
*Server (Attribute)
*User (Attribute)
*Volume (Attribute)

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
*[Root]	Read	All attributes

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The *Host Server* attribute identifies the NCP server that provides the queue management service (QMS) for this queue.

The *Queue Directory* names the subdirectory where this queue's files are stored.

The *Operator*, *Server* and *User* attributes are used by the QMS as access control lists.

The *Operator* attribute identifies users or groups that have operator privileges. The *Server* attribute identifies print servers allowed to service this queue.

The *User* attribute contains a list of objects that are authorized to use this queue. The server that controls the queue must determine if the user list is maintained by an administrator or if the list is automatically generated by the server. If the user list is used by the server as an access control list, the administrator will usually maintain the list. If the user list is purely informational, reflecting access control information stored elsewhere, the server usually maintains the list.

The *See Also* might be used to list related queues. For example, two queues, "Fast" and "Slow," might provide the same set of services, except that "Fast" runs at a higher priority. These two queues might reference each other in their respective *See Also* attributes.

The *Host Resource Name* attribute is used when the host's local identification for a resource differs from the global resource identification. For example, a server might recognize "SYS:" as the local name for a volume with the Directory name

```
"Project X.Engineering.Acme.US"
```

The *L*, *O*, and *OU* attributes are useful when a resource is used by multiple localities, organizations, or organizational units. If these attributes contain appropriate values, a search can be initiated for resources associated with a particular locality or organization.

The *Network Address* attribute (inherited from Resource) acts as a cache for the server's network address. The user can contact the server without having to dereference the *Host Server* attribute.

Computer

Represents both computers that hostNetWare® servers and computers used as client workstations.

NetWare Versions: 4.x

Type: Effective

Super Classes

*Top
Device (Class)

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
*Equivalent To Me (4.1)
*L (Locality)
*Last Referenced Time (4.1)
*Network Address
*O (Organization)
*Obituary
Operator

*OU (Organizational Unit)
*Owner
*Reference
*Revision (4.01)
*See Also
*Serial Number
Server (Attribute)
Status

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The *Operator* attribute is used to list individuals or groups that are responsible for day-to-day maintenance of the computer. This may differ from the value of *Owner*, which can indicate more of an administrative responsibility with respect to the computer.

The *Server* attribute provides a list of servers that are hosted on this computer.

The *See Also* attribute might be used, in this instance, to identify other related computers assigned to a network.

The *L* attribute can be used to identify the physical location of a device. For example, if the device were a printer, the locality might be "Building D, Section 24, by Ed Bender's desk."

The *O* and *OU* might already be present in the device's distinguished name. They are repeated here to aid searching when an organization spans multiple subtrees in the Directory tree. However, these attributes are not added automatically by the Directory even though they may be present in the device's distinguished name. Additional values for the organization or organizational unit may be useful when a device is "co-owned" by multiple organizations.

Country

Defines country entries in the Directory tree. Countries usually appear as immediate subordinates of the root.

NetWare Versions: 4.x

Type: Effective

Super Classes

Top

Containment

Top

Named By

C (Country)

Mandatory Attributes

C (Country)
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
*Equivalent To Me (4.1)
*Last Referenced Time (4.1)
*Obituary
*Reference
*Revision (4.01)

Default ACL Template

Object Name	Default Rights	Affected Attributes

*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The *Description* attribute might contain the full name of the country, since the *C* attribute is restricted to the two letter code defined by ISO 3166.

Device (Class)

Represents physical units that can communicate (such as a modem, printer, and so on). At least one of the *L*, *Serial Number*, or *Owner* attrubutes should be included with the object entry. The choice depends on the type of device.

NetWare Versions: 4.x

Type: Noneffective

Super Classes

Top

Containment

Organization
Organizational Unit

Named By

CN (Common Name)

Mandatory Attributes

CN (Common Name)
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
*Equivalent To Me (4.1)
L (Locality)
*Last Referenced Time (4.1)
Network Address
O (Organization)
*Obituary
OU (Organizational Unit)
Owner

Owner
*Reference
*Revision (4.01)
See Also
Serial Number

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The *L* attribute can be used to identify the physical location of a device. For example, if the device were a printer, the locality might be "Building D, Section 24, by Ed Bender's desk."

The *O* and *OU* may already be present in the device's distinguished name. They are repeated here to aid searching when an organization spans multiple subtrees in the Directory tree. However, these attributes are not added automatically by the Directory even though they may be present in the device's distinguished name. Additional values for the organization or organizational unit may be useful when a device is "co-owned" by multiple organizations.

Directory Map

Represents the physical name of a file system directory path.
NetWare Versions: 4.x
Type: Effective

Super Classes

*Top
Resource (Class)

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
Host Server
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
*Equivalent To Me (4.1)
*Host Resource Name
*L (Locality)
*Last Referenced Time (4.1)
*O (Organization)
*Obituary
*OU (Organizational Unit)

Path (Attribute)
*Reference
*Revision (4.01)
*See Also

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The *Host Server* attribute identifies the server that owns and services the resource. Requests to manipulate a particular resource must usually be directed to the host server.

The *Host Resource Name* attribute is used when the host's local identification for a resource differs from the global resource identification. For example, a server might recognize SYS: as the local name for a volume with the Directory name

```
"Project X.Engineering.Acme.US"
```

The *L*, *O*, and *OU* attributes are useful when a resource is used by multiple localities, organizations, or organizational units. If these attributes contain appropriate values, a search can be initiated for resources associated with a particular locality or organization.

External Entity

Represents a non-native NDS object.
NetWare Versions: 4.1
Type: Effective

Super Classes

Top

Containment

Organization
Organizational Unit

Named By

CN (Common Name)
OU (Organizational Unit)

Mandatory Attributes

CN (Common Name)
*Object Class

OptionalAttributes

*ACL
*Authority Revocation
*Back Link (Attribute)
*Bindery Property
*CA Private Key
*CA Public Key
*Certificate Revocation
*Certificate Validity Interval
*Cross Certificate Pair
Description
EMail Address (Attribute)
*Equivalent To Me (4.1)
External Name
Facsimile Telephone Number (Attribute)
L (Locality)
*Last Referenced Time (4.1)
*Obituary
OU (Organizational Unit)

Physical Delivery Office Name
Postal Address (Attribute)
Postal Code
Postal Office Box
*Reference
*Revision (4.01)
S (State or Province)
SA (Street Address)
See Also
Title

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
[Public]	Read	External Name

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

This object can be used by services that need to store information about entities outside of the Directory. For example, a messaging service that can send messages to E-mail users outside of the Directory needs to store address information about those E-mail users.

A messaging service can use External Entitiy objects to store information about E-mail users who exist on other systems. It can then objects' names in distribution lists that are L ist objects.

Group

Defines values representing an unordered set of names. The names themselves can represent individual objects or other groups of names.

NetWare Versions: 4.x

Type: Effective

Super Classes

Top

Containment

Organization
Organizational Unit

Named By

CN (Common Name)

Mandatory Attributes

CN (Common Name)
Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
*Equivalent To Me (4.1)
Full Name
GID (Group ID)
L (Locality)
*Last Referenced Time (4.1)
Login Script (4.1)
Mailbox ID (4.1)
Mailbox Location (4.1)

Member
O (Organization)
*Obituary
OU (Organizational Unit)
Owner
Profile (Attribute) (4.1)
Profile Membership (4.1)
*Reference
*Revision (4.01)
See Also

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The membership of a group is static; that is, it is explicitly modified by administrative action, rather than dynamically determined each time the group is referred to. The membership of a group can be reduced to a set of individual object's names by replacing each group with its membership. This process can be carried out recursively until all constituent group names have been eliminated, and only the names of individual objects remain.

In general, Directory Service operations do not perform recursive membership expansion. However, access control resolution effectively expands one level of groups listed in an Access Control List (ACL). Thus, if A is a member of group B, which is in turn listed in an ACL, A gains the access granted to group B. However, if A is a member of group B, which is a member of group C, and C is listed in an access control list, A does not gain the access granted to group C.

Other applications are free to perform recursive group expansion, if they so choose.

L, *O*, and *OU* might already be present in the group's distinguished name. They are repeated here to aid searching when an organization spans multiple subtrees in the Directory tree. Additional values for the locality, organization or organizational unit may be useful when a group contains members from multiple organizations, organizational units, or localities.

The *Owner* attribute could be used to contain the name of the group leader or group moderator. This value might not be the same as the set of individuals authorized to modify the group object.

The *See Also* attribute might be used to list related groups. For example, the groups "Project A Programmers", "Project A Writers", and "Project A Testers" might mention one another in their *See Also* attributes.

List

Represents an unordered set of names.
NetWare Versions: 4.1
Type: Effective

Super Classes

Top

Containment

Organization
Organizational Unit

Named By

CN (Common Name)

Mandatory Attributes

CN (Common Name)
Object Class

OptionalAttributes

*ACL
*Authority Revocation
*Back Link (Attribute)
*Bindery Property
*CA Private Key
*CA Public Key
*Certificate Revocation
*Certificate Validity Interval
*Cross Certificate Pair
Description
EMail Address (Attribute)
*Equivalent To Me
Full Name
L (Locality)
*Last Referenced Time
Mailbox ID
Mailbox Location
Member
O (Organization)

*Obituary
OU (Organizational Unit)
Owner
*Reference
*Revision
See Also

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
[Root]	Read	Member

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Member contains the names of the objects that are members of the list. The members can be individual objects (including Group objects) or the names of other List objects.

Unlike Group membership, List membership does not imply security equivalence.

Locality

Defines geographic locations in the Directory tree.
NetWare Versions: 4.x
Type: Effective

Super Classes

Top

Containment

Country
Locality
Organization
Organizational Unit

Named By

L (Locality)
S (State or Province)

Mandatory Attributes

*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
*Equivalent To Me (4.1)
L (Locality)
*Last Referenced Time (4.1)
*Obituary
*Reference
*Revision (4.01)
S (State or Province)

SA (Street Address)
See Also

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

At least one of the attributes *L* and *S* must be present since these are included in the naming rules. It is recommended that only an object contained by a country usea state or a province name as a naming attribute.

Message Routing Group (Class)

Represents a group (or cluster) of messaging servers that have direct connectivity for transferring messages between any two of them.

NetWare Versions: 4.1

Type: Effective

Super Classes

*Top
Group

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
*Object Class

OptionalAttributes

*ACL
*Authority Revocation
*Back Link (Attribute)
*Bindery Property
*CA Private Key
*CA Public Key
*Certificate Revocation
*Certificate Validity Interval
*Cross Certificate Pair
*Description
*EMail Address (Attribute)
*Equivalent To Me
*Full Name
*GID (Group ID)
*L (Locality)
*Last Referenced Time
*Login Script

*Mailbox ID
*Mailbox Location
*Member
*O (Organization)
*Obituary
*OU (Organizational Unit)
*Owner
*Profile (Attribute)
*Profile Membership
*Reference
*Revision
*See Also

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
[Self]	Read	All Attributes
[Self]	Browse	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Member (inherited from Group) is used to list the messaging servers that belong to the Message Routing Group.

Owner (inherited from Group) is used to contain the name of the postmaster general of the messaging server's message routing group. The owner has the authority to add a messaging server's name to, or remove a messaging server's name from the *Member* list.

Messaging Server (Class)

Represents messaging servers (such as NetWare MHS™ servers).

NetWare Versions: 4.1

Type: Effective

Super Classes

*Top
Server (Class)

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
*Object Class

OptionalAttributes

*Account Balance
*ACL
*Allow Unlimited Credit
*Authority Revocation
*Back Link (Attribute)
*Bindery Property
*CA Private Key
*CA Public Key
*Certificate Revocation
*Certificate Validity Interval
*Cross Certificate Pair
*Description
*Equivalent To Me
*Full Name
*Host Device
*L (Locality)
*Last Referenced Time
Message Routing Group (Attribute)

Messaging Database Location
Messaging Server Type
*Minimum Account Balance
*Network Address
*O (Organization)
*Obituary
*OU (Organizational Unit)
Postmaster
*Private Key
*Public Key
*Reference
*Resource (Attribute)
*Revision
*Security Equals
*Security Flags
*See Also
*Status
Supported Services
*User (Attribute)
*Version

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

A MHS messaging server picks up messages which are either submitted by messaging applications (for example, E-mail) or transferred from another messaging server, and delivers them to the recipients. For recipients whose mailboxes are local on the messaging server, the messages are delivered to their mailboxes. Otherwise, the messaging server transfers the messages to another messaging server for eventual delivery to the recipient's mailbox.

A MHS messaging server runs as MHS.NLM on a NetWare server. There is no limit to the number of mailboxes it serves, except that mailboxes take up disk space.

A MHS messaging server is represented by a NDS leaf object whose object class is Messaging Server.

Host Device (inherited from Server) identifies the NCP Server on which the Messaging Server's software runs.

Message Routing Group names the Message Routing Groups to which the Messaging Server is attached.

Messaging Database Location names the volume and path (such as SYS:MHS) on which the message directory resides. MHS messaging servers use a file system subtree to (1) receive messages from servers use a file system subtree to (1) receive messages from applications, other messaging servers and gateways, (2) store messages while they are being routed, (3) store internal control files, and (4) to extract files.

Messaging Server Type identifies the type of the Messaging Server object (for example, MHS, GMHS, X400).

Postmaster specifies one or more users who have the privileges to manage the messaging server, such as privileges to remove a mailbox. Postmasters also receive messages about special events in the messaging server, such as messages being unprocessable.

Supported Services indicates the messaging capabilities of the server.

User (inherited from Server) contains a list users whose mailboxes are serviced by the messaging server. Any effective object that has the *Mailbox ID* and *Mailbox Location* attributes is a valid value for this list.

Adding an object to the *User* list has the same effect as assigning values to the object's *Mailbox Location* and *Mailbox ID* attributes. An administrator can give an object a mailbox by either means.

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
[Public]	Read	Messaging Database Location
[Public]	Read	Messaging Server Type
*[Public]	Read	Network Address
[Self]	Read	All Attributes
[Self]	Browse	[Entry Rights]
*[Self]	Supervisor	[Entry Rights]
*[Self]	Read/Write	Status

NCP Server

Represents servers that provide NCP transport and session services.
NetWare Versions: 4.x
Type: Effective

Super Classes

*Top
Server (Class)

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
*Object Class

Optional Attributes

*Account Balance
*ACL
*Allow Unlimited Credit
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
DS Revision (4.1)
*Equivalent To Me (4.1)
*Full Name
*Host Device
*L (Locality)
*Last Referenced Time (4.1)

Messaging Server (Attribute) (4.1)
*Minimum Account Balance
*Network Address
*O (Organization)
*Obituary
Operator
*OU (Organizational Unit)
*Private Key
*Public Key
*Reference
*Resource (Attribute)
*Revision (4.01)
*Security Equals (4.1)
*Security Flags (4.1)
*See Also
*Status
Supported Services
*User (Attribute)
*Version

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
[Public]	Read	Messaging Server
*[Public]	Read	Network Address
*[Self]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Note that the individual services on an NCP server do not need distinct Directory names, since they can all share a common NCP session. However, individual resources on a server may require distinct Directory entries. For example, if a server supports file services and queue management services, that server will have only one object in the Directory for the server itself. Other Directory entries would denote the individual queues and file volumes.

The *Supported Services* attribute is used to list NCP based services or features available at this network address. It should not be used to list other services residing on the same host.

The *Operator* attribute is used by the NCP server as an ACL. If an object is listed in this attribute, that object is allowed to perform remote-console operations.

NCP Server is intended to represent both Bindery based and NDS based NCP servers. The *Version* attribute (inherited from Server) should distinguish one type of server from the other

The *Private Key* and *Public Key* attributes are present if the server is a client of the Directory'sAuthentication Services. The *Resource* attribute contains a list of resources managed by this service.

The *User* attribute contains a list of objects that are authorized to use this server. The server must determine if the user list is to be maintained by an administrator, or if the list is automatically generated by the server. If the user list is used by the server as an ACL, the administrator usually maintains the list. If the user list is purely informational, reflecting access control information stored elsewhere, the server usually maintains the list.

Organization

Defines organization objects in the Directory tree.
NetWare Versions: 4.x
Type: Effective

Super Classes

Top

Containment

Country
Locality
Top

Named By

O (Organization)

Mandatory Attributes

O (Organization)
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
Detect Intruder
EMail Address (Attribute)
*Equivalent To Me (4.1)
Facsimile Telephone Number (Attribute)
Intruder Attempt Reset Interval
Intruder Lockout Reset Interval
L (Locality)
*Last Referenced Time (4.1)

Lockout After Detection
Login Intruder Limit
Login Script
Mailbox ID (4.1)
Mailbox Location (4.1)
NNS Domain
*Obituary
Physical Delivery Office Name
Postal Address (Attribute)
Postal Code
Postal Office Box
Print Job Configuration
Printer Control
*Reference
*Revision (4.01)
S (State or Province)
SA (Street Address)
See Also
Telephone Number (Attribute)

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

An organization located directly under the root denotes an international organization. For international organizations, the values of the *O* attribute must all be distinct.

Organizational Person

Defines objects representing people employed by, or in some other important way associated with an organization.

NetWare Versions: 4.x

Type: Noneffective

Super Classes

*Top
Person

Containment

Organization
Organizational Unit

Named By

CN (Common Name)
OU (Organizational Unit)

Mandatory Attributes

*CN (Common Name)
*Object Class
*Surname

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
EMail Address (Attribute)
*Equivalent To Me (4.1)
Facsimile Telephone Number (Attribute)
*Full Name
*Generational Qualifier (4.1)

*Given Name (4.1)
*Initials (4.1)
L (Locality)
*Last Referenced Time (4.1)
Mailbox ID (4.1)
Mailbox Location (4.1)
*Obituary
OU (Organizational Unit)
Physical Delivery Office Name
Postal Address (Attribute)
Postal Code
Postal Office Box
*Reference
*Revision (4.01)
S (State or Province)
SA (Street Address)
*See Also
*Telephone Number (Attribute)
Title

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The X.500 standard defines two subclasses of person: Organizational Person and Residential Person. The schema defined by this document does not include Residential Person, but the division of Person from Organizational Person has been maintained for future compatibility with X.500.

Organizational Role

Defines a position or role within an organization.
NetWare Versions: 4.x
Type: Effective

Super Classes

Top

Containment

Organization
Organizational Unit

Named By

CN (Common Name)

Mandatory Attributes

CN (Common Name)
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
EMail Address (Attribute)
*Equivalent To Me (4.1)
Facsimile Telephone Number (Attribute)
L (Locality)
*Last Referenced Time (4.1)
Mailbox ID (4.1)
Mailbox Location (4.1)
*Obituary
OU (Organizational Unit)

Physical Delivery Office Name
Postal Address (Attribute)
Postal Code
Postal Office Box
*Reference
*Revision (4.01)
Role Occupant
S (State or Province)
SA (Street Address)
See Also
Telephone Number (Attribute)

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Normally, an organizational role is thought to be performed by a particular organizational person. Over its lifetime, however, an organizational role may be filled by a succession of different organizational people. In general, an organizational role may be filled by a person or a nonhuman entity.

Organizational Unit

Defines objects representing subdivisions of organizations.
NetWare Versions: 4.x
Type: Effective

Super Classes

Top

Containment

Locality
Organization
Organizational Unit

Named By

OU (Organizational Unit)

Mandatory Attributes

*Object Class
OU (Organizational Unit)

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
Detect Intruder
EMail Address (Attribute)
*Equivalent To Me (4.1)
Facsimile Telephone Number (Attribute)
Intruder Attempt Reset Interval
Intruder Lockout Reset Interval
L (Locality)
*Last Referenced Time (4.1)

Lockout After Detection
Login Intruder Limit
Login Script
Mailbox ID (4.1)
Mailbox Location (4.1)
NNS Domain
*Obituary
Physical Delivery Office Name
Postal Address (Attribute)
Postal Code
Postal Office Box
Print Job Configuration
Printer Control
*Reference
*Revision (4.01)
S (State or Province)
SA (Street Address)
See Also
Telephone Number (Attribute)

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
[Self]	Read	Print Job Configuration
[Self]	Read	Login Script

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Partition

Encapsulates the information required to maintain the synchronization and connectivity of the Directory's distributed operation.

NetWare Versions: 4.x

Type: Noneffective

Super Classes

Top

Containment

(none)

Named By

(none)

Mandatory Attributes

*Object Class

Optional Attributes

*ACL
Authority Revocation
*Back Link (Attribute)
*Bindery Property
CA Private Key
CA Public Key
Certificate Revocation
Convergence
Cross Certificate Pair
*Equivalent To Me (4.1)
High Convergence Sync Interval
Inherited ACL
*Last Referenced Time (4.1)
Low Convergence Reset Time
Low Convergence Sync Interval
*Obituary
Partition Control (4.01)
Partition Creation Time
Received Up To
*Reference

Replica
Replica Up To (4.1)
*Revision (4.01)
Synchronized Up To

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The name server automatically adds this class (and its required attributes) to the root object of a partition. (This class is added in addition to the base class of the object, but does not change that base class.) Most of the partition attributes are operational in nature, so the name server can supply initial values for these attributes automatically.

Person

Represents the common elements of organizational and residential persons.
NetWare Versions: 4.x
Type: Noneffective

Super Classes

Top

Containment

(none)

Named By

(none)

Mandatory Attributes

CN (Common Name)
*Object Class
Surname

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
*Equivalent To Me (4.1)
Full Name
Generational Qualifier (4.1)
Given Name (4.1)
Initials (4.1)
*Last Referenced Time (4.1)
*Obituary
*Reference
*Revision (4.01)

See Also
Telephone Number (Attribute)

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The X.500 standard defines two subclasses of person: Organizational Person and Residential Person. The schema defined by this document does not include Residential Person, but the division of Person from Organizational Person has been maintained for future compatibility with X.500.

Print Server (Class)

Represents NetWare print servers.
NetWare Versions: 4.x
Type: Effective

Super Classes

*Top
Server (Class)

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
*Object Class

Optional Attributes

*Account Balance
*ACL
*Allow Unlimited Credit
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
*Equivalent To Me (4.1)
*Full Name
*Host Device
*L (Locality)
*Last Referenced Time (4.1)
*Minimum Account Balance

*Network Address
*O (Organization)
*Obituary
Operator
*OU (Organizational Unit)
Printer (Attribute)
*Private Key
*Public Key
*Reference
*Resource (Attribute)
*Revision (4.01)
SAP Name
*Security Equals (4.1)
*Security Flags (4.1)
*See Also
*Status
*User (Attribute)
*Version

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
*[Public]	Read	Network Address
[Root]	Read	All Attribute Rights
*[Self]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Print Server differs from the NCP Server class in that print servers use the SPX protocol for communications rather than NCP.

The Print Server class is used for both Bindery based and NDS based print servers. Bindery based print servers do have the *Public Key* or *Private Key* attributes. The *Version* attribute (inherited from Server) indicates whether the server is Bindery based or NDS based.

The *Operator* and *User* attributes are used by the print server as access control lists. (The *User* attribute is inherited from Server.) The *Operator* attribute identifies those individuals who are authorized to act as print server operators. The *User* attribute identifies individuals authorized to use the print server.

The *Queue* attribute lists the queues that are serviced by this print server.

The *Host Device* attribute identifies the device that hosts the server. This is usually a computer, but might be some other device. For example, a printer could host a built-in print server.

The *Private Key* and *Public Key* attributes are present if the server is a client of the Directory's Authentication Services. The *Resource* attribute contains a list of resources managed by this service.

The *User* attribute contains a list of objects that are authorized to use this server. The server must determine if the user list is to be maintained by an administrator, or if the list is automatically generated by the server. If the user list is used by the server as an access control list, the administrator will usually maintain the list. If the user list is purely informational, reflecting access control information stored elsewhere, the server usually maintains the list.

Printer (Class)

Represents printers in the Directory tree. A printer object points to the queues to which it is attached.

NetWare Versions: 4.x

Type: Effective

Super Classes

*Top
Device (Class)

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
Cartridge
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Default Queue
*Description
*Equivalent To Me (4.1)
Host Device
*L (Locality)
*Last Referenced Time (4.1)
Memory

Network Address
Network Address Restriction
Notify
*O (Organization)
*Obituary
Operator
*OU (Organizational Unit)
*Owner
Page Description Language
Print Server (Attribute)
Printer Configuration
Queue (Attribute)
*Reference
*Revision (4.01)
*See Also
*Serial Number
Status
Supported Typefaces

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The *Host Device* attribute is used in this class to denote the computer (or other device) to which the printer is attached.

The *Host Server* attribute identifies the print servers that manage this device.

The *Supported Typefaces* and *Page Description Language* attributes are included to aid a search for printers with a particular set of capabilities. The contents of these attributes are statically maintained by an administrator, rather than being dynamically updated from printer feedback.

The *Queue* attribute identifies the associated queues through which this printer may be accessed.

The *L* attribute can be used to identify the physical location of a device. For example, if the device were a printer, the locality might be "Building

D, Section 24, by Ed Bender's desk."

The organization name and organizational unit name may already be present in the device's distinguished name. They are repeated here to aid searching when an organization spans multiple subtrees in the Directory tree. However, these attributes are not added automatically by the Directory even though they may be present in the device's distinguished name. Additional values for the organization name or organizational unit name may be useful when a device is "co-owned" by multiple organizations.

Profile (Class)

Specifies a shared login configuration.
NetWare Versions: 4.x
Type: Effective

Super Classes

Top

Containment

Organization
Organizational Unit

Named By

CN (Common Name)

Mandatory Attributes

CN (Common Name)
Login Script
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
*Equivalent To Me (4.1)
Full Name (4.1)
L (Locality)
*Last Referenced Time (4.1)
O (Organization)
*Obituary
OU (Organizational Unit)
*Reference

*Revision (4.01)
See Also

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

A profile has an associated login script that contains the bulk of the configuration information.

Queue (Class)

Represents batch processing queues available in the NetWare NCP environment.

NetWare Versions: 4.x

Type: Effective

Super Classes

*Top
Resource (Class)

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
*Object Class
Queue Directory

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
Device (Attribute)
*Equivalent To Me (4.1)
*Host Resource Name
Host Server
*L (Locality)
*Last Referenced Time (4.1)

Network Address
*O (Organization)
*Obituary
Operator
*OU (Organizational Unit)
*Reference
*Revision (4.01)
*See Also
Server (Attribute)
User (Attribute)
Volume (Attribute)

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
[Root]	Read	All attributes

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The *Host Server* attribute identifies the server that owns and services the resource. Requests to manipulate a particular resource must usually be directed to the host server.

The *Host Resource Name* attribute is used when the host's local identification for a resource differs from the global resource identification. For example, a server might recognize SYS: as the local name for a volume with the Directory name

```
"Project X.Engineering.Acme.US"
```

The *L*, *O*, and *OU* attributes are useful when a resource is used by multiple localities, organizations, or organizational units. If these attributes contain appropriate values, a search can be initiated for resources associated with a particular locality or organization.

The *Network Address* attribute acts as a cache for the server's network address. The user can contact the server without having to dereference the *Host Server* attribute.

The *User* attribute contains a list of objects that are authorized to use this resource. The server that controls the resource must determine if the user list is maintained by an administrator or if the list is automatically generated by the server. If the user list is used by the server as an access control list, the administrator will usually maintain the list. If the user list is purely informational, reflecting access control information stored elsewhere, the server usually maintains the list.

Resource (Class)

Identifies logical resources available on the network.
NetWare Versions: 4.x
Type: Noneffective

Super Classes

Top

Containment

Organization
Organizational Unit

Named By

CN (Common Name)

Mandatory Attributes

CN (Common Name)
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
*Equivalent To Me (4.1)
Host Resource Name
L (Locality)
*Last Referenced Time (4.1)
O (Organization)
*Obituary
OU (Organizational Unit)
*Reference
*Revision (4.01)

See Also

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The Resource class differs from the Device class in that a device is a physical unit, and a resource is some nonphysical, logical unit. Examples of resources are queues, profiles, and file system volumes.

The *Host Resource Name* attribute is used when the host's local identification for a resource differs from the global resource identification. For example, a server might recognize SYS: as the local name for a volume with the Directory name

```
"Project X.Engineering.Acme.US"
```

The *L*, *O*, and *OU* attributes are useful when a resource is used by multiple localities, organizations, or organizational units. If these attributes contain appropriate values, a search can be initiated for resources associated with a particular locality or organization.

Server (Class)

Identifies entities that manage one or more resources and provide access to those resources through a communications protocol.

NetWare Versions: 4.x

Type: Noneffective

Super Classes

Top

Containment

Organization
Organizational Unit

Named By

CN (Common Name)

Mandatory Attributes

CN (Common Name)
*Object Class

Optional Attributes

Account Balance
*ACL
Allow Unlimited Credit
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
Description
*Equivalent To Me (4.1)
Full Name
Host Device
L (Locality)
*Last Referenced Time (4.1)
Minimum Account Balance

Network Address
O (Organization)
*Obituary
OU (Organizational Unit)
Private Key
Public Key
*Reference
Resource (Attribute)
*Revision (4.01)
Security Equals (4.1)
Security Flags (4.1)
See Also
Status
User (Attribute)
Version

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
[Public]	Read	Network Address
[Self]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

The *Host Device* attribute identifies the device that hosts the server. This is usually a computer, but might be some other device. For example, a printer could host a built-in print server.

The *Private Key* and *Public Key* attributes are present if the server is a client of the Directory's Authentication Services. The *Resource* attribute contains a list of resources managed by this service.

The *User* attribute contains a list of objects that are authorized to use this server. The server must determine if the user list is to be maintained by an administrator, or if the list is automatically generated by the server. If the user list is used by the server as an access control list, the administrator will usually maintain the list. If the user list is purely informational, reflecting access control information stored elsewhere, the server usually maintains the list.

Top

All classes are a subclass of Top. This class mandates that all objects contain an *Object Class* attribute. Although Top is an effective class, it is a special case in that no objects can be constructed from this class by the user.

NetWare Versions: 4.x

Type: Effective

Super Classes

(none)

Containment

(none)

Named By

(none)

Mandatory Attributes

Object Class

Optional Attributes

ACL
Authority Revocation (4.1)
Back Link (Attribute)
Bindery Property
CA Private Key (4.1)
CA Public Key (4.1)
Certificate Revocation (4.1)
Certificate Validity Interval (4.1)
Cross Certificate Pair (4.1)
Equivalent To Me (4.1)
Last Referenced Time (4.1)
Obituary
Reference
Revision (4.01)

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]

*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Unknown (Class)

Represents any object created by the server to restore an object whose base class is no longer defined by the Schema.

NetWare Versions: 4.x

Type: Effective

Super Classes

Top

Containment

(none)

Named By

(none)

Mandatory Attributes

Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Equivalent To Me (4.1)
*Last Referenced Time (4.1)
*Obituary
*Reference
*Revision (4.01)

Default ACL Template

Object Name	**Default Rights**	**Affected Attributes**
*[Creator]	Supervisor	[Entry Rights]

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

User (Class)

Represents users of network services.
NetWare Versions: 4.x
Type: Effective

Super Classes

*Top
*Person
Organizational Person

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)
*OU (Organizational Unit)

Mandatory Attributes

*CN (Common Name)
*Object Class
*Surname

Optional Attributes

Account Balance
*ACL
Allow Unlimited Credit
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
*EMail Address (Attribute)
*Equivalent To Me (4.1)
*Facsimile Telephone Number (Attribute)

Full Name
*Generational Qualifier (4.1)
*Given Name (4.1)
Group Membership
Higher Privileges
Home Directory
*L (Locality)
Language
Last Login Time
*Last Referenced Time (4.1)
Locked By Intruder
Login Allowed Time Map
Login Disabled
Login Expiration Time
Login Grace Limit
Login Grace Remaining
Login Intruder Address
Login Intruder Attempts
Login Intruder Reset Time
Login Maximum Simultaneous
Login Script
Login Time
*Mailbox ID (4.1)
*Mailbox Location (4.1)
Message Server
Minimum Account Balance
Network Address
Network Address Restriction
*Obituary
*OU (Organizational Unit)
Password Allow Change
Password Expiration Interval
Password Expiration Time
Password Minimum Length
Password Required
Password Unique Required
Passwords Used
*Physical Delivery Office Name
*Postal Address (Attribute)
*Postal Code
*Postal Office Box
Print Job Configuration
Printer Control
Private Key
Profile (Attribute)

Profile Membership (4.1)
Public Key
*Reference
*Revision (4.01)
*S (State or Province)
*SA (Street Address)
Security Equals
*See Also
Server Holds
*Telephone Number (Attribute)
*Title
UID (User ID)

Default ACL Template

Object Name	Default Rights	Affected Attributes
*[Creator]	Supervisor	[Entry Rights]
[Public]	Read	Message Server
[Root]	Browse	[Entry Rights]
[Root]	Read	Group Membership
[Root]	Read	Network Address
[Self]	Read	All attributes
[Self]	Read/Write	Login Script
[Self]	Read/Write	Print Job Configuration

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

Note that User includes both clients and service providers. In this context, *Private Key* stores the object's private key encrypted by the object's password.

The X.500 standard defines two subclasses of person: Organizational Person and Residential Person. The schema defined by this document does not include Residential Person, but the division of Person from Organizational Person has been maintained for future compatibility with X.500.

Volume (Class)

Represents NetWare file system volumes.
NetWare Versions: 4.x
Type: Effective

Super Classes

*Top
Resource (Class)

Containment

*Organization
*Organizational Unit

Named By

*CN (Common Name)

Mandatory Attributes

*CN (Common Name)
Host Server
*Object Class

Optional Attributes

*ACL
*Authority Revocation (4.1)
*Back Link (Attribute)
*Bindery Property
*CA Private Key (4.1)
*CA Public Key (4.1)
*Certificate Revocation (4.1)
*Certificate Validity Interval (4.1)
*Cross Certificate Pair (4.1)
*Description
*Equivalent To Me (4.1)
*Host Resource Name
*L (Locality)
*Last Referenced Time (4.1)
*O (Organization)
*Obituary
*OU (Organizational Unit)

*Reference
*Revision (4.01)
*See Also
Status

Default ACL Template

Object Name	**Default Rights**	**Affected Attributes**
*[Creator]	Supervisor	[Entry Rights]
[Root]	Read	Host Resource Name
[Root]	Read	Host Server

Remarks

For help in understanding the class definition template, see Reading Class Definitions.

When present, the *Host Resource Name* attribute (inherited from Resource) is used to contain the local volume name that corresponds to the volume name on the server. If the attribute is not present, the local volume name "SYS:" can be assumed.

This subclass exists primarily to allow Volume objects to be distinguished from other types of Resource objects.

The *Host Server* attribute identifies the server that owns and services the resource. Requests to manipulate a particular resource must usually be directed to the host server.

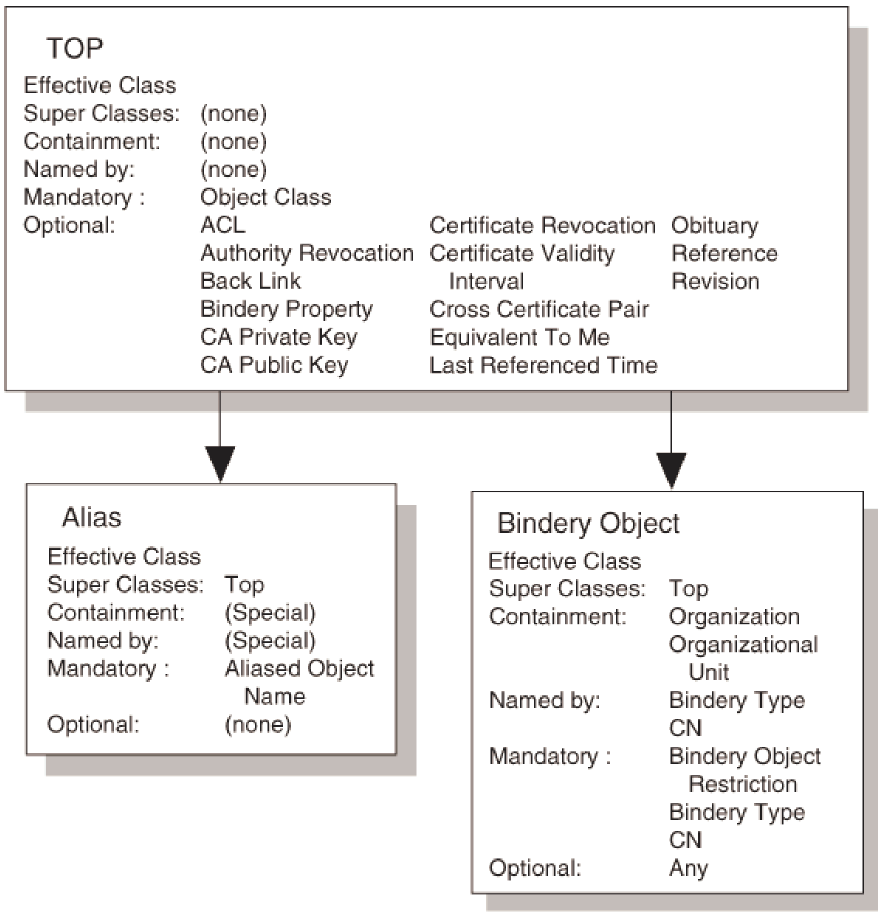
The *Host Resource Name* attribute is used when the host's local identification for a resource differs from the global resource identification. For example, a file server might recognize SYS: as the local name for a volume with the Directory name

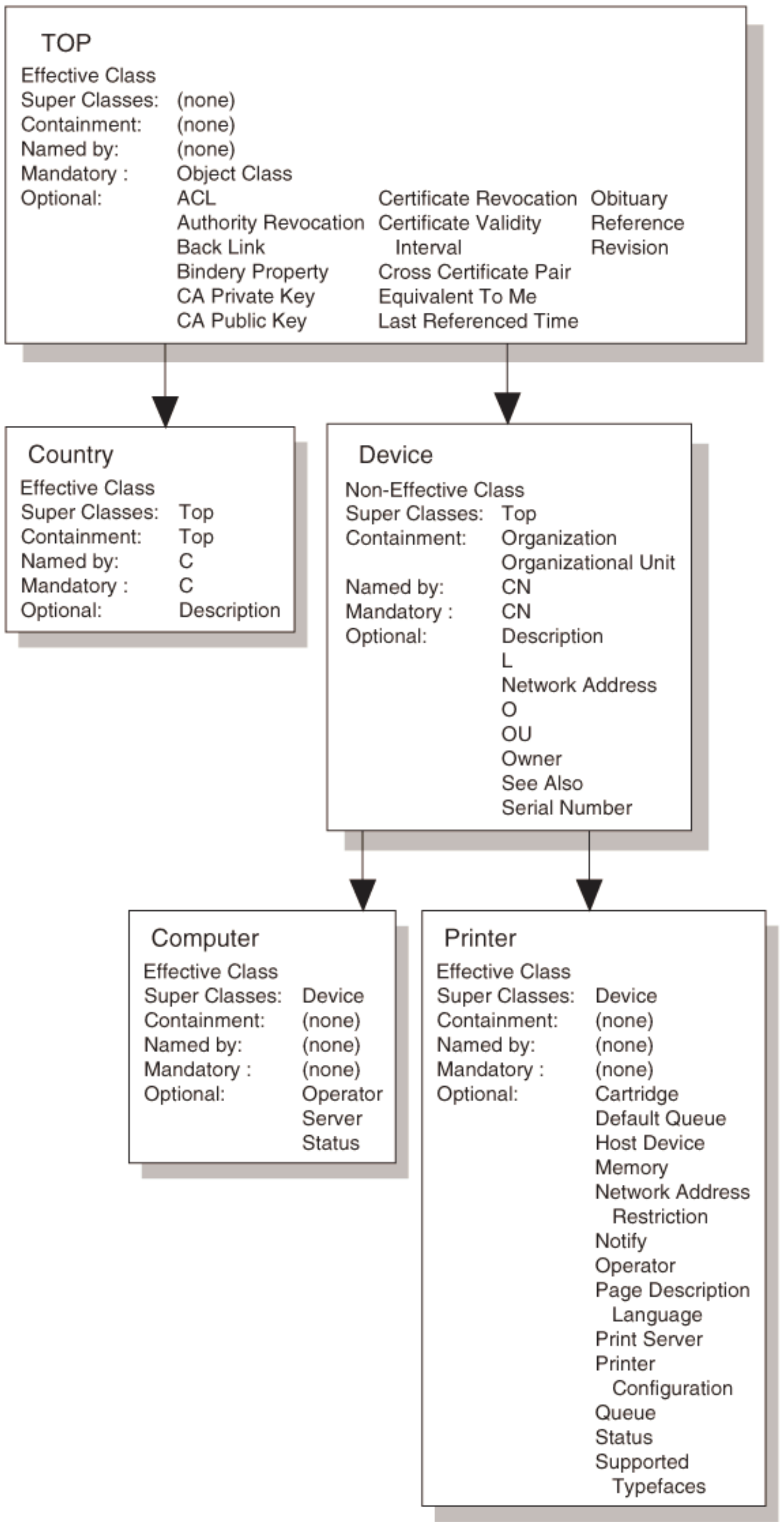
```
"Project X.Engineering.Acme.US"
```

The *L*, *O*, and *OU* attributes are useful when a resource is used by multiple localities, organizations, or organizational units. If these attributes contain appropriate values, a search can be initiated for resources associated with a particular locality or organization.

Graphical View of NDS Object Class Definitions

For more information see Graphical View Explanation.





TOP

Effective Class

Super Classes:	(none)
Containment:	(none)
Named by:	(none)
Mandatory :	Object Class
Optional:	ACL, Authority Revocation, Back Link, Bindery Property, CA Private Key, CA Public Key, Certificate Revocation, Certificate Validity Interval, Cross Certificate Pair, Equivalent To Me, Last Referenced Time, Obituary, Reference, Revision

Country

Effective Class

Super Classes:	Top
Containment:	Top
Named by:	C
Mandatory :	C
Optional:	Description

Device

Non-Effective Class

Super Classes:	Top
Containment:	Organization, Organizational Unit
Named by:	CN
Mandatory :	CN
Optional:	Description, L, Network Address, O, OU, Owner, See Also, Serial Number

Computer

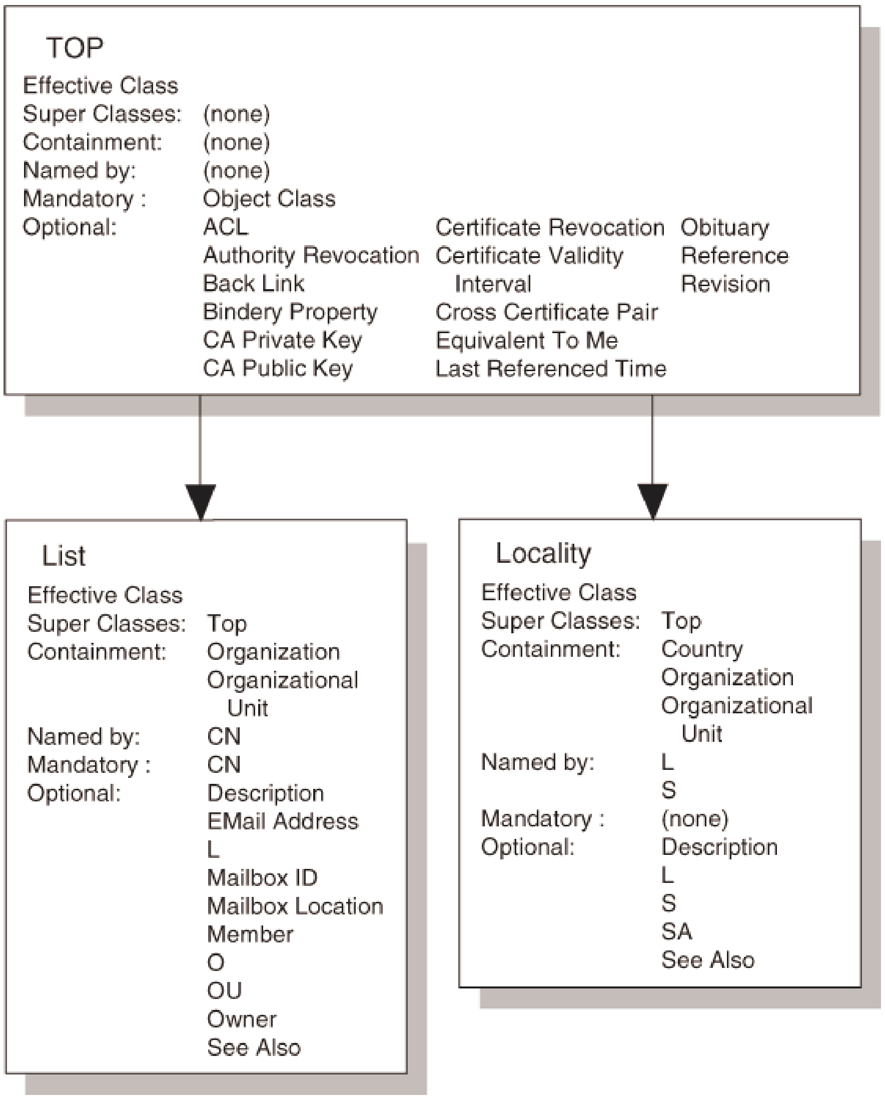
Effective Class

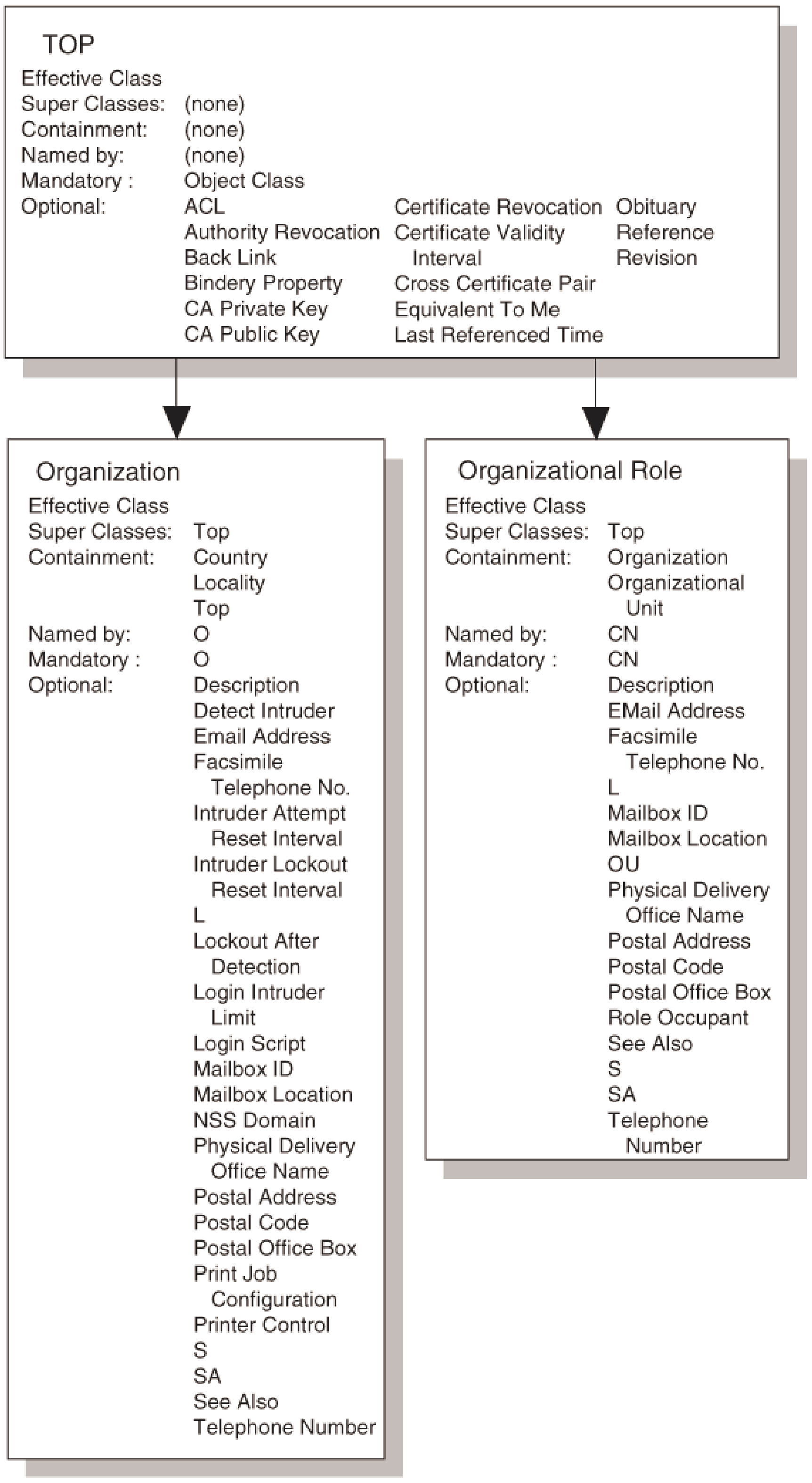
Super Classes:	Device
Containment:	(none)
Named by:	(none)
Mandatory :	(none)
Optional:	Operator, Server, Status

Printer

Effective Class

Super Classes:	Device
Containment:	(none)
Named by:	(none)
Mandatory :	(none)
Optional:	Cartridge, Default Queue, Host Device, Memory, Network Address Restriction, Notify, Operator, Page Description Language, Print Server, Printer Configuration, Queue, Status, Supported Typefaces


TOP
Effective Class
Super Classes: (none)
Containment: (none)
Named by: (none)
Mandatory : Object Class
Optional: ACL
Authority Revocation
Back Link
Bindery Property
CA Private Key
CA Public Key
Certificate Revocation
Certificate Validity Interval
Cross Certificate Pair
Equivalent To Me
Last Referenced Time
Obituary
Reference
Revision
List
Effective Class
Super Classes: Top
Containment: Organization
Organizational Unit
Named by: CN
Mandatory : CN
Optional: Description
EMail Address
L
Mailbox ID
Mailbox Location
Member
O
OU
Owner
See Also
Locality
Effective Class
Super Classes: Top
Containment: Country
Organization
Organizational Unit
Named by: L
S
Mandatory : (none)
Optional: Description
L
S
SA
See Also



TOP

Effective Class

Super Classes:	(none)
Containment:	(none)
Named by:	(none)
Mandatory :	Object Class
Optional:	ACL Authority Revocation Back Link Bindery Property CA Private Key CA Public Key Certificate Revocation Certificate Validity Interval Cross Certificate Pair Equivalent To Me Last Referenced Time Obituary Reference Revision

Organization

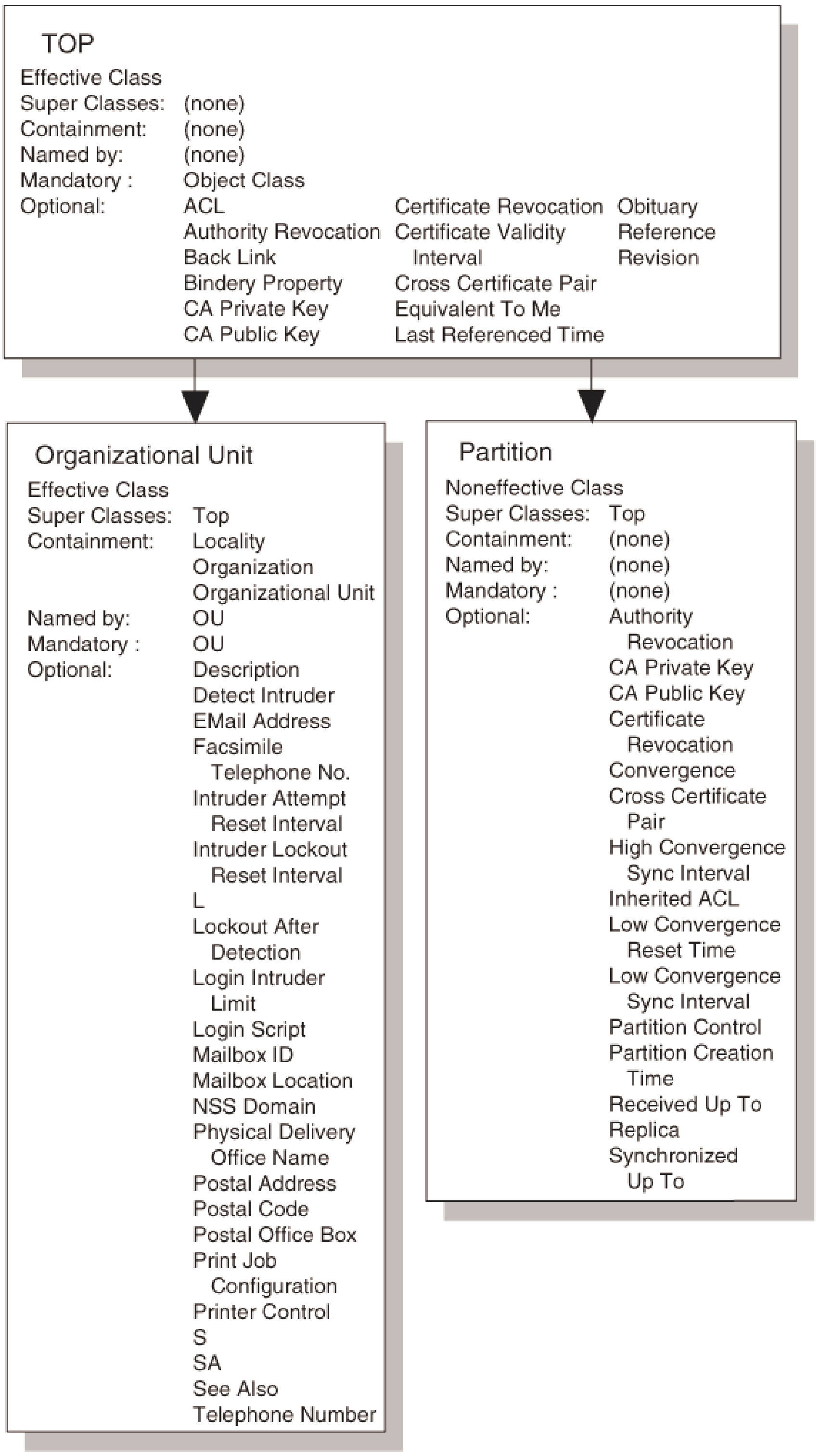
Effective Class

Super Classes:	Top
Containment:	Country Locality Top
Named by:	O
Mandatory :	O
Optional:	Description Detect Intruder Email Address Facsimile Telephone No. Intruder Attempt Reset Interval Intruder Lockout Reset Interval L Lockout After Detection Login Intruder Limit Login Script Mailbox ID Mailbox Location NSS Domain Physical Delivery Office Name Postal Address Postal Code Postal Office Box Print Job Configuration Printer Control S SA See Also Telephone Number

Organizational Role

Effective Class

Super Classes:	Top
Containment:	Organization Organizational Unit
Named by:	CN
Mandatory :	CN
Optional:	Description EMail Address Facsimile Telephone No. L Mailbox ID Mailbox Location OU Physical Delivery Office Name Postal Address Postal Code Postal Office Box Role Occupant See Also S SA Telephone Number



TOP

Effective Class

Super Classes:	(none)		
Containment:	(none)		
Named by:	(none)		
Mandatory :	Object Class		
Optional:	ACL	Certificate Revocation	Obituary
	Authority Revocation	Certificate Validity Interval	Reference
	Back Link	Cross Certificate Pair	Revision
	Bindery Property	Equivalent To Me	
	CA Private Key	Last Referenced Time	
	CA Public Key		

Organizational Unit

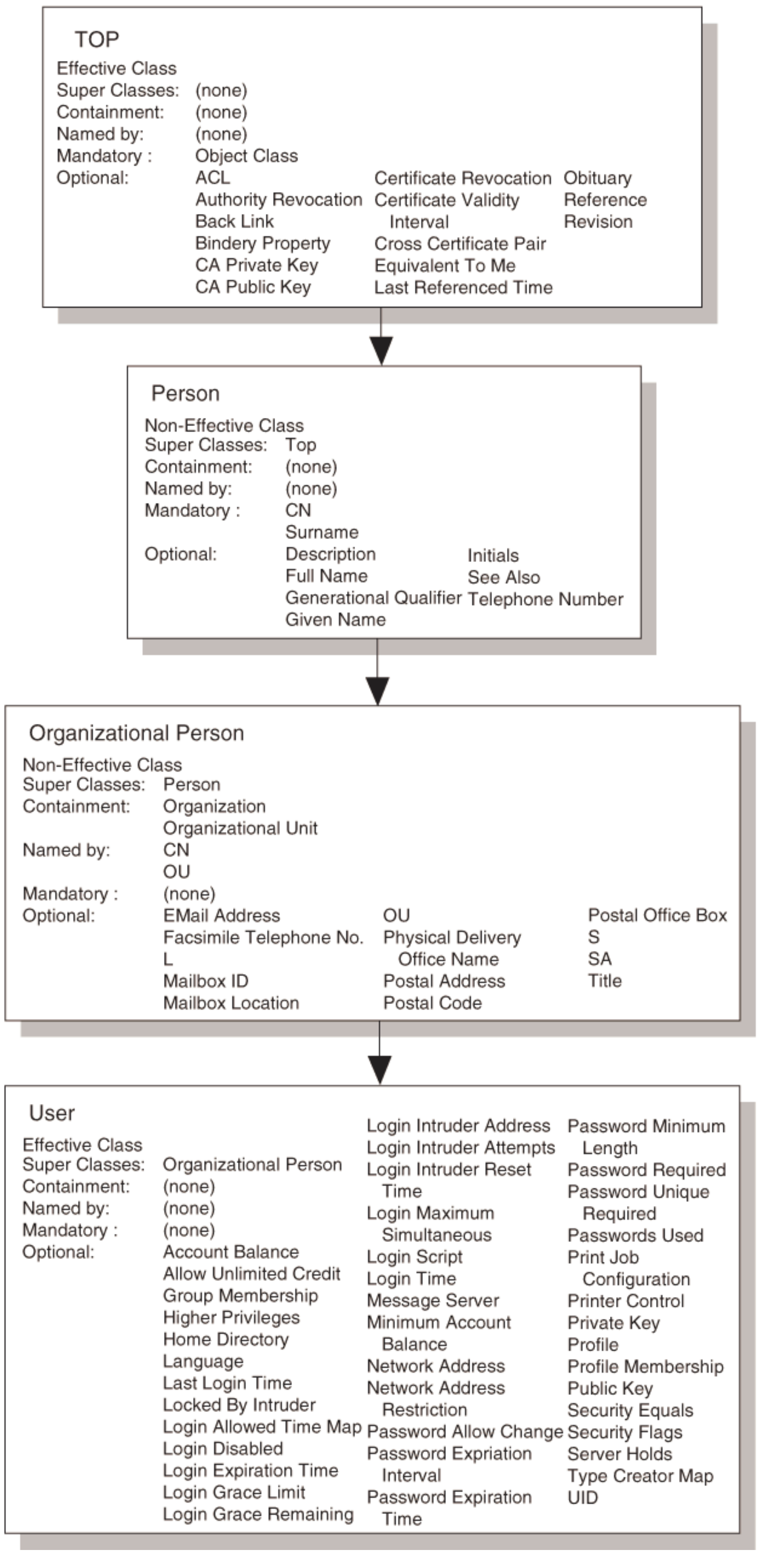
Effective Class

Super Classes:	Top
Containment:	Locality Organization Organizational Unit
Named by:	OU
Mandatory :	OU
Optional:	Description Detect Intruder EMail Address Facsimile Telephone No. Intruder Attempt Reset Interval Intruder Lockout Reset Interval L Lockout After Detection Login Intruder Limit Login Script Mailbox ID Mailbox Location NSS Domain Physical Delivery Office Name Postal Address Postal Code Postal Office Box Print Job Configuration Printer Control S SA See Also Telephone Number

Partition

Noneffective Class

Super Classes:	Top
Containment:	(none)
Named by:	(none)
Mandatory :	(none)
Optional:	Authority Revocation CA Private Key CA Public Key Certificate Revocation Convergence Cross Certificate Pair High Convergence Sync Interval Inherited ACL Low Convergence Reset Time Low Convergence Sync Interval Partition Control Partition Creation Time Received Up To Replica Synchronized Up To

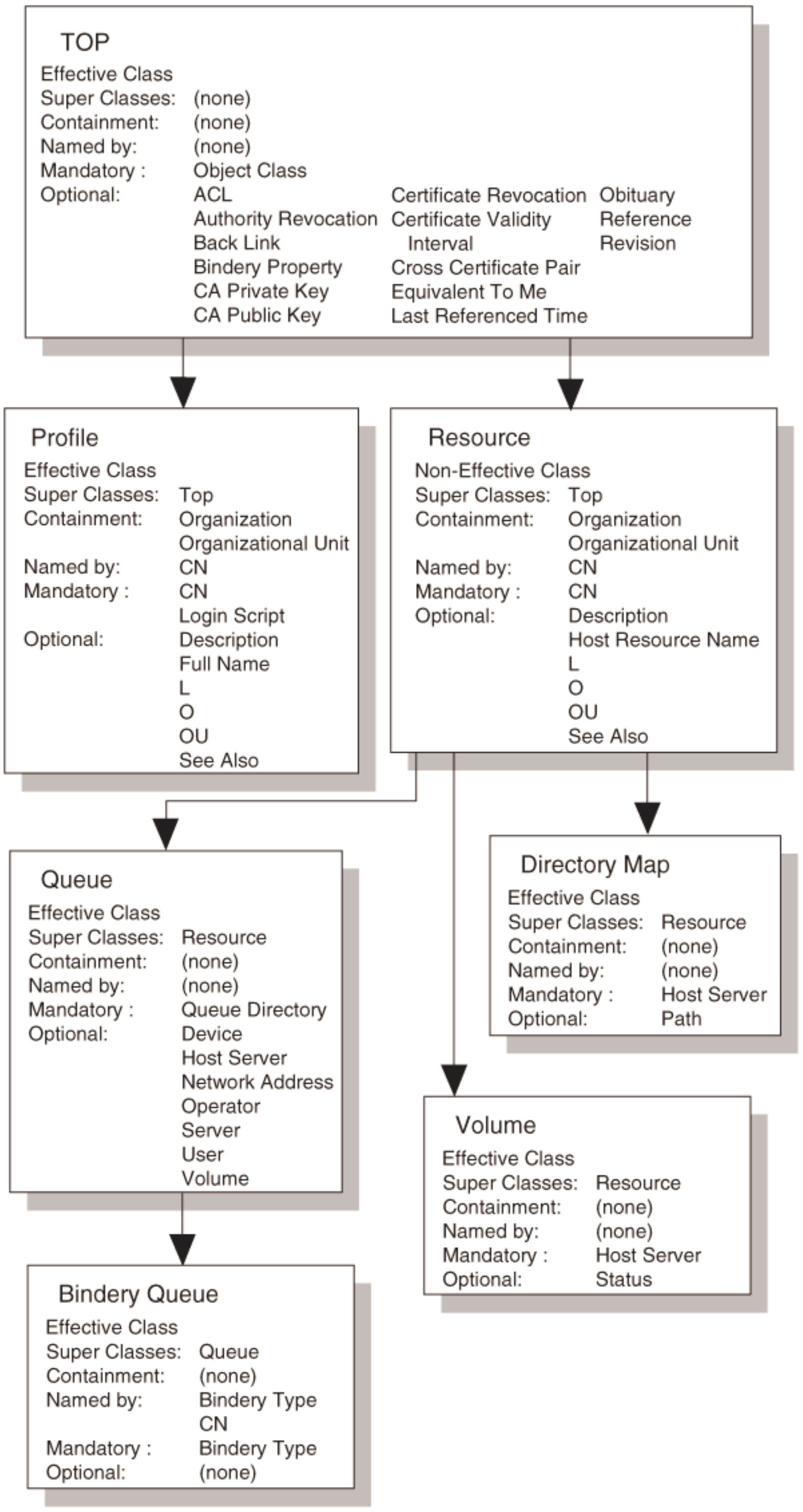


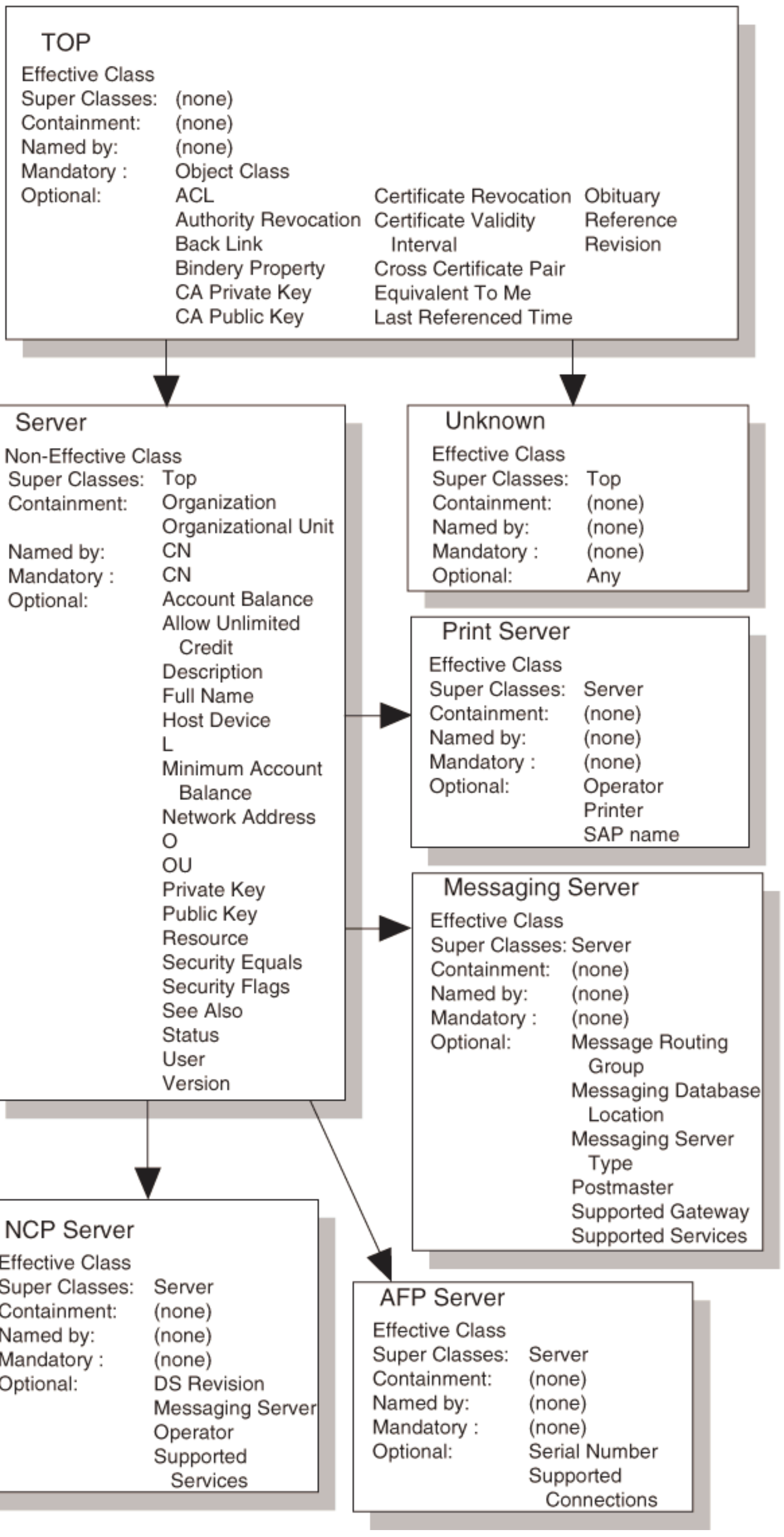
TOP
Effective Class
Super Classes: (none)
Containment: (none)
Named by: (none)
Mandatory : Object Class
Optional: ACL
Authority Revocation
Back Link
Bindery Property
CA Private Key
CA Public Key
Certificate Revocation
Certificate Validity Interval
Cross Certificate Pair
Equivalent To Me
Last Referenced Time
Obituary
Reference
Revision

Person
Non-Effective Class
Super Classes: Top
Containment: (none)
Named by: (none)
Mandatory : CN
Surname
Optional: Description
Full Name
Generational Qualifier
Given Name
Initials
See Also
Telephone Number

Organizational Person
Non-Effective Class
Super Classes: Person
Containment: Organization
Organizational Unit
Named by: CN
OU
Mandatory : (none)
Optional: EMail Address
Facsimile Telephone No.
L
Mailbox ID
Mailbox Location
OU
Physical Delivery Office Name
Postal Address
Postal Code
Postal Office Box
S
SA
Title

User
Effective Class
Super Classes: Organizational Person
Containment: (none)
Named by: (none)
Mandatory : (none)
Optional: Account Balance
Allow Unlimited Credit
Group Membership
Higher Privileges
Home Directory
Language
Last Login Time
Locked By Intruder
Login Allowed Time Map
Login Disabled
Login Expiration Time
Login Grace Limit
Login Grace Remaining
Login Intruder Address
Login Intruder Attempts
Login Intruder Reset Time
Login Maximum Simultaneous
Login Script
Login Time
Message Server
Minimum Account Balance
Network Address
Network Address Restriction
Password Allow Change
Password Expriation Interval
Password Expiration Time
Password Minimum Length
Password Required
Password Unique Required
Passwords Used
Print Job Configuration
Printer Control
Private Key
Profile
Profile Membership
Public Key
Security Equals
Security Flags
Server Holds
Type Creator Map
UID


TOP
Effective Class
Super Classes: (none)
Containment: (none)
Named by: (none)
Mandatory : Object Class
Optional: ACL
Authority Revocation
Back Link
Bindery Property
CA Private Key
CA Public Key
Certificate Revocation
Certificate Validity Interval
Cross Certificate Pair
Equivalent To Me
Last Referenced Time
Obituary
Reference
Revision
Profile
Effective Class
Super Classes: Top
Containment: Organization
Organizational Unit
Named by: CN
Mandatory : CN
Login Script
Optional: Description
Full Name
L
O
OU
See Also
Resource
Non-Effective Class
Super Classes: Top
Containment: Organization
Organizational Unit
Named by: CN
Mandatory : CN
Optional: Description
Host Resource Name
L
O
OU
See Also
Queue
Effective Class
Super Classes: Resource
Containment: (none)
Named by: (none)
Mandatory : Queue Directory
Optional: Device
Host Server
Network Address
Operator
Server
User
Volume
Directory Map
Effective Class
Super Classes: Resource
Containment: (none)
Named by: (none)
Mandatory : Host Server
Optional: Path
Volume
Effective Class
Super Classes: Resource
Containment: (none)
Named by: (none)
Mandatory : Host Server
Optional: Status
Bindery Queue
Effective Class
Super Classes: Queue
Containment: (none)
Named by: Bindery Type
CN
Mandatory : Bindery Type
Optional: (none)


TOP
Effective Class
Super Classes: (none)
Containment: (none)
Named by: (none)
Mandatory : Object Class
Optional: ACL
Authority Revocation
Back Link
Bindery Property
CA Private Key
CA Public Key
Certificate Revocation
Certificate Validity Interval
Cross Certificate Pair
Equivalent To Me
Last Referenced Time
Obituary
Reference
Revision
Server
Non-Effective Class
Super Classes: Top
Containment: Organization
Organizational Unit
Named by: CN
Mandatory : CN
Optional: Account Balance
Allow Unlimited Credit
Description
Full Name
Host Device
L
Minimum Account Balance
Network Address
O
OU
Private Key
Public Key
Resource
Security Equals
Security Flags
See Also
Status
User
Version
Unknown
Effective Class
Super Classes: Top
Containment: (none)
Named by: (none)
Mandatory : (none)
Optional: Any
Print Server
Effective Class
Super Classes: Server
Containment: (none)
Named by: (none)
Mandatory : (none)
Optional: Operator
Printer
SAP name
Messaging Server
Effective Class
Super Classes: Server
Containment: (none)
Named by: (none)
Mandatory : (none)
Optional: Message Routing Group
Messaging Database Location
Messaging Server Type
Postmaster
Supported Gateway
Supported Services
NCP Server
Effective Class
Super Classes: Server
Containment: (none)
Named by: (none)
Mandatory : (none)
Optional: DS Revision
Messaging Server
Operator
Supported Services
AFP Server
Effective Class
Super Classes: Server
Containment: (none)
Named by: (none)
Mandatory : (none)
Optional: Serial Number
Supported Connections

Related Topics:

NDS Object Class Definitions

NDS Attribute Type Definitions

Account Balance

Specifies the amount of credit (or money) the user has to spend on the purchase of network services. If the user's account balance drops below a specified minimum balance, services are refused to the user.

NetWare Versions: 4.x

Syntax

Counter

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Server (Class)
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

ACL

Contains access control information for the object and its attributes.
NetWare Versions: 4.x

Syntax

Object ACL

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Aliased Object Name

The *Aliased Object Name* attribute is assigned to alias objects in the Directory. The aliased object is the object to which the alias points.

NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Alias

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Allow Unlimited Credit

Allows the user to access and use all network services he or she has rights to, without maintaining a minimum account balance.

NetWare Versions: 4.x

Syntax

Boolean

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Server (Class)
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Authority Revocation

A time-stamped list of revoked public keys of all Certification Authorities known and certified by the Certification Authority.

NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition
Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Back Link (Attribute)

Attached to any object for which an external reference is required by a remote server.

NetWare Versions: 4.x

Syntax

Back Link (Syntax)

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SERVER_READ

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute contains the set of servers that store an external reference to an associated object. The attribute is used to notify such servers of changes in the status of the object.

Bindery Object Restriction

A single-valued integer attribute used by Bindery objects. It consists of an error code that indicates the reason the Bindery object cannot be represented as a Directory object.

NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR

Used In

Bindery Object

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Bindery Property

Emulates bindery properties that cannot be represented with other attribute types.
NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

In the bindery, properties of any name and data structure could be attached to objects. This is not the case with the Directory. Bindery Property attributes are used to hold the information stored in bindery properties through the bindery API and the Bindery Services.

Bindery Type

Associates a bindery object type with an object of class Bindery Object.
NetWare Versions: 4.x

Syntax

Numeric String

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_STRING_ATTR

Used In

Bindery Object
Bindery Queue

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

C (Country)

Specifies a country.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (2, 2)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Country

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

When used as a component of a Directory name, a country name identifies the country in which the named object is physically located or with which it is associated in some other important way. An attribute value for country name is a string chosen from ISO 3166.

CA Private Key

(Certification Authority Private Key) Contains the certification authority private key.

NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_HIDDEN_ATTR
DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE

Used In

Partition
Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Your applications cannot access this attribute since it is a hidden attribute.

This attribute is used to sign public keys which are produced for objects. The attribute contains the private key encrypted with the certification authority's password.

CA Public Key

(Certification Authority Public Key) Contains the certification authority public key.
NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE

Used In

Partition
Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute is used to verify public keys that are produced for objects subordinate to the certification authority. The attribute contains the public key along with the certification information.

Cartridge

Contains a list of font cartridges present on the printer.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Certificate Revocation

A time-stamped list of all public keys revoked by the Certification Authority.

NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition
Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Certificate Validity Interval

Specifies the amount of time that a certificate is valid.
NetWare Versions: 4.1

Syntax

Interval

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (60..-1)
DS_SYNC_IMMEDIATE (4.1)

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

The length of time is specified when the certificate is made.

CN (Common Name)

Specifies an identifier of an object. A common name is not a complete Directory name; it is a name by which an object is commonly known in a particular context, such as within an organization.

NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Bindery Object
Bindery Queue
Device (Class)
External Entity
Group
List
Organizational Person
Organizational Role
Person
Profile (Class)
Resource (Class)
Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute is a string chosen by the person or organization it describes. For devices and application entities, the string is chosen by the organization responsible for the object. A common name is not necessarily definitive and may admit ambiguity within its limited scope.

Common names conform to the naming conventions of the country or culture with which they are associated. For example, a typical common name for a person in an English-speaking country may comprise a personal title (Mr., Ms., Dr., Professor, Sir, Lord, and so on), a first name, middle names, a last name, a generational qualifier (Jr., Sr., and so on) and decorations and awards. Examples of common names include:

```
CN=Mr Robin Lachlan Mcleod BSc(Hons) CEng MIEE
CN=Divisional Coordination Committee
CN=High Speed Modem
```

Variant names are associated with a named object as separate and alternative attribute values. Common variant names should be made available (for example, the use of a middle name as a preferred first name or the use of "Bill" in place of "William").

Convergence

Indicates how persistent a partition should be in attempting to keep its replicas up to date.
NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (0,1)
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

The possible values of this attribute and their meanings are:

	0	Low	Do not propagate updates as they come in. Rather, wait for a periodic synchronization to update any replicas. Synchronizations may be done at a low frequency to save resources. Partitions with low convergence and pending updates should be synchronized at least once every 24 hours. See the Low Convergence Sync Interval and Low Convergence Reset Time attributes for details on setting the convergence cycle.
	1	High	Make one attempt to propagate an update to all replicas when it comes in. If this fails, schedule a synchronization for the next interval time.

High is the default for this attribute.

Cross Certificate Pair

A pair of public keys that allow public key verification to circumvent the normal certification hierarchy. This provides a shorter certification path.

NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition
Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Default Queue

Specifies a queue where jobs submitted to the specified printer go unless a different queue is specified.

NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SERVER_READ
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Description

Specifies text that describes the associated object.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..1024)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Country
Device (Class)
External Entity
Group
List
Locality
Organization
Organizational Role
Organizational Unit
Person
Profile (Class)
Resource (Class)
Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

An attribute value for the *Description* attribute is a string. For example, the object "Standards Interest" might have the associated description "distribution list for exchange of information about intracompany standards development".

Detect Intruder

Indicates a desire to identify suspicious login attempts.
NetWare Versions: 4.x

Syntax

Boolean

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Unit

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Device (Attribute)

A list of all printers that service the specified queue.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Queue (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

DS Revision

Contains the internal revision number of the NDS™ agent that is running on a NetWare® server.

NetWare Versions: 4.1

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE

Used In

NCP Server

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

EMail Address (Attribute)

Contains the EMail address of the user. The name must conform to the established conventions for e-mail names.

NetWare Versions: 4.x

Syntax

EMail Address (Syntax)

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ (4.1)
DS_SYNC_IMMEDIATE (4.1)

Used In

External Entity
Group
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

A user can have only one E-mail address. The E-mail address for a MHS user is stored in the Mailbox Location and Mailbox ID attributes. The E-mail address for a non-MHS user is stored in the EMail Address attribute.

EMail Address is used to specify a user's mailbox that resides in a non-MHS E-mail system. For example, an NDS user can chose to have his or her E-mail delivered to a UNIX machine which supports the SMTP messaging protocol. This user's SMTP address in the UNIX machine is placed in the EMail Address attribute. (In this case, the user's Mailbox Location and Mailbox ID attributes are not used.)

EMail Address can also specify a user's E-mail alias as known in a foreign messaging system. A non-MHS messaging system can use the alias to send mail to a MHS user. For example, a MHS user (whose mailbox is in a MHS messaging server) can have an X400 alias so that X400 users can use this alias to send mail to the MHS user.

A user can have more than one E-mail alias, one for each non-MHS system.

The EMail Address attribute information is stored in an EMail_Address_T structure. The convention for storing non-MHS E-mail addresses and aliases in the structure is as follows:

If the *type* field of the EMail_Address_T structure is set to zero, the data structure contains an E-mail address, in the form of **non-MHS_Email_Protocol:non-MHS_Email_Address**. Where **non_MHS_Email_Protocol** is a 1-8 character string, and **non-MHS_Email_Address** is a string for the actual address value.

Example: SMTP:JohnD@Novell.Com

If the *type* field of the EMail_Address_T structure is set to one, the data structure contains an E-mail alias, in the form of **non-MHS_Email_Protocol:non-MHS_Email_Alias**. Where **non_MHS_Email_Protocol** is a 1-8 character string, and **non-MHS_Email_Address** is a string for the actual alias value.

Example: SMTP:JohnD@Novell.Com

External Name

Specifies the name of the external entity in the form used by that service.
NetWare Versions: 4.1

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE

Used In

External Entity

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

The *External Name* attribute is used to hold names that will not be interpreted by the Directory. These names should not be translated.

MHS uses *External Name* to include users from not-NDS directories, in order to provide an integrated address book for sending mail.

Equivalent To Me

Specifies a list of objects that are security equivalent to the object containing the attribute.

NetWare Versions: : 4.1

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SERVER_READ
DS_SYNC_IMMEDIATE

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

External Synchronizer

Reserved for future use.
NetWare Versions: 4.1

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE

Used In

(none)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Facsimile Telephone Number (Attribute)

Specifies the telephone number and, optionally, the parameters for a facsimile terminal associated with an object.
NetWare Versions: 4.x

Syntax

Facsimile Telephone Number (Syntax)

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

External Entity
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

An attribute value for the facsimile telephone number is a string that complies with the internationally agreed format for showing international telephone numbers, E.123, (for example "+81 3 347 7418") and an optional bit string (formatted according to Recommendation T.30).

Full Name

Specifies the full name of an object.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (0..127)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Group
List
Person
Profile (Class)
Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Generational Qualifier

Specifies the generation of an object.
NetWare Versions: 4.1

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..8)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE

Used In

Person

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Junior, **Jr.**, **Senior**, **Sr.**, and **II** are examples of generational qualifiers.

GID (Group ID)

Specifies a unique group ID for use by UNIX® clients.
NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Group

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Given Name

Specifies the given name of an object.
NetWare Versions: 4.1

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ (4.1)
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..32)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE

Used In

Person

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

In the American culture, *Given Name* is the first name of a person. It does not include the family name.

Group Membership

Contains a list of the groups to which the object belongs.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE
DS_WRITE_MANAGED

Used In

External Entity
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

High Convergence Sync Interval

Contains the interval at which a partition synchronization will occur if no events have caused synchronization to occur.

NetWare Versions: 4.x

Syntax

Interval

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

When high convergence is in effect, synchronization of partition information is event-driven. When a partition's information is changed, an immediate synchronization with other replicas is initiated. If no update activity occurs within the high convergence synchronization interval, synchronization will be initiated. The default for this interval is 60 minutes.

Higher Privileges

Specifies an alternative set of security access privileges.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SERVER_READ
DS_SYNC_IMMEDIATE
DS_WRITE_MANAGED

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute is not currently implemented. Its purpose is to allow a person to activate a certain set of privileges, then deactivate them when desired. This avoids the need to log in as supervisor to perform certain functions. You can activate the higher privileges, perform the desired function, and then deactivate the privileges.

Home Directory

Contains the initial value for a client's current working directory.
NetWare Versions: 4.x

Syntax

Path (Syntax)

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..255)
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

The *Home Directory* attribute is intended for use primarily by UNIX clients. DOS and OS/2* clients use login scripts to provide their initial configuration information, and MAC clients have other means for storing configuration information. UNIX clients maintain their configuration information in the client's home directory. The *Home Directory* attribute should contain a legal name in the UNIX name space of the Host Server defined for the client.

Host Device

Contains the distinguished name of the device with which the object is associated.

NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)
Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Host Resource Name

Contains the name by which the resource is known on the local host.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Resource (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute is used when a resource is locally known by a different name on the host server. For example, a volume's global name might be:

```
'Tools.Development.Acme.US'
```

That volume's local name on the server might be SYS:.

Host Server

Identifies the server with which an object is associated.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Directory Map
Queue (Class)
Volume (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Inherited ACL

Contains a summary of access control information inherited from a superior partition.
NetWare Versions: 4.x

Syntax

Object ACL

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

The information in this attribute is used to optimize effective rights calculations within a partition. The attribute lists all access control information inherited from objects superior to the partition object.

Initials

Specifies the initials of an object.
NetWare Versions: 4.1

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..8)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE

Used In

Person

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

For example, the initials for John Steven Doe are JSD.

Intruder Attempt Reset Interval

Designates the time frame in which to monitor consecutive failed login attempts.
NetWare Versions: 4.x

Syntax

Interval

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Unit

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

If the number of consecutive failed attempts to log in exceeds the *Login Intruder Limit*, the user is locked from attempting further logins. The *Intruder Attempt Reset Interval* is the amount of time in which those consecutive login attempts must fall. For example, assume the interval is 30 minutes and the limit is 3 login attempts. If a user attempted to log in every 20 minutes, the user would never be locked out. If the user attempted to log in every minute, a lock would be implemented after the third attempt.

Intruder Lockout Reset Interval

Identifies the amount of time a user remains locked out once a lock has been applied.

NetWare Versions: 4.x

Syntax

Interval

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Unit

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

L (Locality)

Specifies a physical or geographical location.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..128)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Device (Class)
External Entity
Group
List
Locality
Organization
Organizational Person
Organizational Role
Organizational Unit
Profile (Class)
Resource (Class)
Server (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

When used as a component of a Directory name, a locality name identifies a geographical area or locality in which the named object is physically located or with which it is associated in some other important way. An attribute value for *L* is a string, (for example, L = Edinburgh).

Language

Contains an ordered list of languages.
NetWare Versions: 4.x

Syntax

Case Ignore List

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

This attribute is intended for use by applications to determine the language display desired by the object.

Last Login Time

Contains the login time of the session previous to the current session.
NetWare Versions: 4.x

Syntax

Time

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Last Referenced Time

Specifies the last time the object was referenced.
NetWare Versions: 4.1

Syntax

Timestamp

Constraints

DS_NONREMOVABLE_ATTR
DS_SF_PER_REPLICA
DS_SINGLE_VALUED_ATTR

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Locked By Intruder

Specifies that an object has been disabled due to intruder detection.
NetWare Versions: 4.x

Syntax

Boolean

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

In the case of a User object, if TRUE this attribute indicates the user cannot log in to the network because too many sequential attempts to log in have been denied. This usually happens when a person does not know the correct password and tries to guess it too many times.

Lockout After Detection

Indicates that users should be kept from attempting to log in once they are identified as an intruder.

NetWare Versions: 4.x

Syntax

Boolean

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Unit

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Monitoring attempts of users to log in may occur without actually enforcing any restrictions on logging in. This attribute may be used to indicate that users should be locked out once the limits of tolerance have been exceeded. Those tolerable limits are designated using other attribute values.

Login Allowed Time Map

Specifies the allowed login time periods for an account for each day of the week to a precision of one-half hour.

NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (42, 42)
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Disabled

Informs the user that the account has been disabled. This can be because of Supervisor disabling, intruder detection, no more grace logins, and so on.

NetWare Versions: 4.x

Syntax

Boolean

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Expiration Time

Specifies a date and time after which a client cannot log in and authenticate as an object.

NetWare Versions: 4.x

Syntax

Time

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Grace Limit

Specifies the total number of times an old password can be used (after the old password has expired) to access the account.

NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Grace Remaining

Specifies how many grace logins are left before the account is locked.
NetWare Versions: 4.x

Syntax

Counter

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Intruder Address

Specifies the address of the node that caused the intruder detection lockout.
NetWare Versions: 4.x

Syntax

Net Address

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Intruder Attempts

Specifies the number of failed login attempts.
NetWare Versions: 4.x

Syntax

Counter

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Intruder Limit

Specifies the number of failed login attempts allowed before an account is locked due to intruder detection.

NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Unit

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Intruder Reset Time

Specifies the next time that the intruder attempts variable will be reset.
NetWare Versions: 4.x

Syntax

Time

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Maximum Simultaneous

Specifies the number of simultaneous login sessions authenticated for the object.

NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Login Script

Contains the Login Script.
NetWare Versions: 4.x

Syntax

Stream

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Unit
Profile (Class)
User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

The *Login Script* attribute replaces the system login script. When a user logs in, the LOGIN program searches one level above (to either the Organization or Organizational Unit) and runs its script (if any), then runs the user's login script.

Login Time

Specifies the login time of the current session.
NetWare Versions: 4.x

Syntax

Time

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Low Convergence Reset Time

Specifies the time at which to start the low convergence synchronization cycle.
NetWare Versions: 4.x

Syntax

Time

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

This reset time is implemented only if low convergence has been specified in the *Convergence* attribute of the partition. The time stored in this attribute is the time of day at which a synchronization occurs and the interval counter is reset (or initialized). The next synchronization will occur after the low synchronization interval has lapsed.

Low Convergence Sync Interval

Specifies the amount of time (in seconds) that must pass from the start of one partition synchronization to the next.

NetWare Versions: 4.x

Syntax

Interval

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

This synchronization interval is only implemented if low convergence has been set (in the *Convergence* attribute).

Mailbox ID

Contains a unique ID associated with the object's mailbox and messaging server.

NetWare Versions: 4.1

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..8)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE

Used In

Group
List
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Mailbox Location

Contains the name of the Messaging Server that services the object's mailbox.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE

Used In

Group
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Member

Lists objects associated with a group or list.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Group
List

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Memory

Specifies the amount of printer memory (in kilobytes).
NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Message Routing Group (Attribute)

Specifies the name of the Message Routing Groups to which a messaging server can belong.

NetWare Versions: 4.1

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE

Used In

Messaging Server (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Message Server

Specifies the name of a server object that stores and forwards broadcast-type messages.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Messaging Database Location

Contains the location of the messaging file-directory structure.
NetWare Versions: 4.1

Syntax

Path (Syntax)

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE

Used In

Messaging Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

An example for a messaging directory is "\mhs".

MHS messaging servers use a file-system subtree to (1) receive messages from applications, other messaging servers and gateways, (2) store messages while they are being routed, (3) store internal control files, and (4) to extract files.

Messaging Server (Attribute)

Identifies a Messaging Server that is running on the NCP Server.
NetWare Versions: 4.1

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE

Used In

NCP Server

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Messaging Server Type

Identifies the type of a Messaging Server.
NetWare Versions: 4.1

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..32)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE

Used In

Messaging Server (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Examples of messaging server types are MHS, GMHS, and X400.

Minimum Account Balance

Specifies the minimum amount of credit (or money) a user must have in his or her account to access specified services.

NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Server (Class) (4.1)
User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Network Address

Contains one or more network addresses of the associated object.
NetWare Versions: 4.x

Syntax

Net Address

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Device (Class)
Queue (Class)
Server (Class)
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute is an octet string and an integer that denotes the type of the underlying transport. The attribute can recur.

Network Address Restriction

Restricts objects to specific network or node addresses.
NetWare Versions: 4.x

Syntax

Net Address

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

NNS Domain

Specifies the name of the NNS Domain that has been upgraded into the container.

NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1.128)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Unit

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Notify

Specifies a list of objects that are to be notified of a specified event.
NetWare Versions: 4.x

Syntax

Typed Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

O (Organization)

Specifies an organization.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Device (Class)
Group
List
Organization
Profile (Class)
Resource (Class)
Server (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

When used as a component of a Directory name, an organization name identifies an organization with which the named object is affiliated. An attribute value for *O* is a string chosen by the organization (for example, O=Scottish Telecommunications plc). Any variant's name should be associated with the named organization as separate and alternative attribute values.

Obituary

Used to avoid name collisions during certain operations.
NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_HIDDEN_ATTR
DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Your applications cannot access this attribute since it is a hidden attribute.

This attribute is used internally by NDS. Certain circumstances can require a unique identification other than the object name in order to resolve certain functions. When an object is renamed, for example, an obituary time stamp is recorded. Updates to the object might have occurred elsewhere but have not yet been synchronized on the partition where the rename occurred. When the skulking reaches the partition with the renamed object, it is not be able to find the renamed object by name. The update eventually takes place through use of the time stamp as identification in the obituary attribute.

Object Class

Contains an unordered list of object classes. These classes are the fully expanded set of super classes for the object to which this attribute is assigned.

NetWare Versions: 4.x

Syntax

Class Name

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

When an object is created, a single initial value for object class must be specified. When the server creates the object, it expands the value set of the object class attribute to include all of the super classes of the initially specified class.

Operator

Specifies an object with operator privileges.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SERVER_READ
DS_SYNC_IMMEDIATE (4.1)

Used In

Computer
NCP Server
Print Server (Class)
Printer (Class)
Queue (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

OU (Organizational Unit)

Specifies an organizational unit.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Device (Class)
External Entity (4.1)
Group
List
Organizational Person
Organizational Role
Organizational Unit
Profile (Class)
Resource (Class)
Server (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

When used as a component of a Directory name, an organizational unit name identifies a unit of an organization with which the named object is affiliated. The unit is understood to be part of an organization designated by an *O* attribute assigned to the same object. It follows that if an *OU* attribute is used in a Directory name, it must be associated with an *O* attribute.

An attribute value for *O* is a string chosen by the organization to which the unit belongs. An example of an organizational unit name is:

```
OU=Technology Division
```

If "TD" is a common abbreviation for Technology Division, it must be assigned as a separate, alternative attribute value (OU = TD).

Owner

Specifies the name of an object that has some responsibility for the associated object.

NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Device (Class)
Group
List

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

An attribute value for this attribute is a distinguished name (which could represent a group), and can recur.

Page Description Language

Identifies the page description languages (PDLs) supported by a printer. Multiple PDLs should be represented as multiple values.

NetWare Versions: 4.x

Syntax

Printable String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

An example of an attribute value for this attribute is:

```
Page Description Language=PostScript
```

Partition Control

Contains the states of split and join operations
NetWare Versions: 4.x

Syntax

Typed Name

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_READ_ONLY_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Partition Creation Time

Identifies the particular incarnation of the list of replicas of a partition.
NetWare Versions: 4.x

Syntax

Timestamp

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Partition Creation Times are defined by explicit management intervention to recover from hard failures of replicas that prevent synchronizations from completing.

Password Allow Change

Determines whether the person logged in under an account can change the password for that account.

NetWare Versions: 4.x

Syntax

Boolean

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Password Expiration Interval

Specifies the time interval after which passwords expire.
NetWare Versions: 4.x

Syntax

Interval

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Password Expiration Interval uses a syntax definition (SYN_INTERVAL) that is a 32 bit integer. The time is stored in seconds. NWADMIN divides the value by 86400 (seconds per day) and displays it in days.

Password Expiration Time

Specifies the next time that a password will expire.
NetWare Versions: 4.x

Syntax

Time

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Password Minimum Length

Establishes the minimum length for an object's clear-text password.
NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Password Required

Establishes that a password is required for the object to log in.
NetWare Versions: 4.x

Syntax

Boolean

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Password Unique Required

Establishes that when an object's password is changed, it must be unique (different) from those in the *Passwords Used* attribute.

NetWare Versions: 4.x

Syntax

Boolean

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Passwords Used

Specifies old (previously used) passwords.
NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_HIDDEN_ATTR
DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Your applications cannot access this attribute since it is a hidden attribute.

Path (Attribute)

Specifies the physical location of a file system directory.
NetWare Versions: 4.x

Syntax

Path (Syntax)

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR

Used In

Directory Map

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Physical Delivery Office Name

Specifies the name of the city, village, and so on, where a physical delivery office is situated.

NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..128)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

External Entity
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Postal Address (Attribute)

Specifies the address information required for the physical delivery of postal messages to a named object.
NetWare Versions: 4.x

Syntax

Postal Address (Syntax)

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

External Entity
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

A value for this attribute is typically composed of selected attributes from the MHS Unformatted Postal O/R Address version 1 according to Recommendation F.401. The address is limited to 6 lines of 30 characters each, including a Postal Country Name.

Normally the information contained in such an address could include an addressee's name, street address, city, state or province, postal code and possibly a Post Office Box number depending on the specific requirements of the named object.

NetWare administrative utilities will link the postal address value to the following attributes: *Physical Delivery Office Name*, *Postal Code*, *Postal Office Box*, and *Street Address*.

The postal address uses the following as default values:

Line 1: The object's RDN

Line 2: Street Address or Post Office Box Number

Line 3: (no default value)

Line 4: Physical Delivery Office Name, State or Province Name

Line 5: Postal Code

Line 6: Country Name (from the object's DN)

The second line of the postal address will default to the post office box, if present, or otherwise to the street address. The third line of the postal address should be presented in the utility as the second line of both the post office box and street address. Each line of the postal address is limited to 30 characters in length. Attribute values are truncated to 30 characters when used in the postal address. Developers are encouraged to follow these guidelines.

Postal Code

Specifies the postal code of the named object. If this attribute value is present, it is part of the object's postal address.

NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (0..40)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

External Entity
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Postal Office Box

Specifies the post office box at which the object receives physical postal delivery.

NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (0..40)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

External Entity (4.1)
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Postmaster

Specifies one or more users who have the privileges to manage a messaging server, such as privileges to remove a mailbox.

NetWare Versions: 4.1

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE

Used In

Messaging Server (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Postmasters also receive messages about special events in the messaging server, such as messages being unprocessable.

Print Job Configuration

Contains information on the specified print job configuration.
NetWare Versions: 4.x

Syntax

Stream

Constraints

DS_NONREMOVABLE_ATTR
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Unit
User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Print Server (Attribute)

Designates an object as the host server for a specific printer.
NetWare Versions: 4.x

Syntax

Typed Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Printer (Attribute)

Contains a list of object names of printers that are serviced by the print server.

NetWare Versions: 4.x

Syntax

Typed Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Print Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Printer Configuration

Contains information on the specified printer configuration.
NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Printer Control

The NDS™ counterpart of the DOS printer definition file NET$PRN.DAT.
NetWare Versions: 4.x

Syntax

Stream

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Unit
User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Private Key

Contains an RSA private key.
NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_HIDDEN_ATTR
DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE

Used In

Server (Class)
User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Your applications cannot access this attribute since it is a hidden attribute.

This attribute is used to produce signed authentication information that can be used to verify the identity of an object.

The *Private Key* attribute contains an RSA private key encrypted with the password of the object.

Profile (Attribute)

Identifies the login profile to be used if the user doesn't specify one at login time.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Group (4.1)
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Profile Membership

Contains a list of profiles that the object can use.
NetWare Versions: 4.1

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE
DS_WRITE_MANAGED

Used In

Group
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute allows objects to have multiple profiles. This is useful for NNS users who are brought into the Directory, since NNS allows users to have multiple domains.

Public Key

Contains a certified RSA public key.
NetWare Versions: 4.x

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE

Used In

Server (Class)
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute is used to verify authentication information produced with the corresponding private key.

The attribute contains the public key, along with information that certifies the integrity of the public/private key pair. Certification information is necessary to prevent legitimate key pairs from being replaced by illicit key pairs.

Queue (Attribute)

Contains the distinguished name of the queue with which the object is associated.

NetWare Versions: 4.x

Syntax

Typed Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Queue Directory

Contains the name of the subdirectory where the queue's files are stored.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SERVER_READ
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..255)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Queue (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

The subdirectory name should include the local volume designation along with the subdirectory name. The subdirectory name should be a valid DOS name.

Received Up To

Specifies the last time the replica has received any updates.
NetWare Versions: 4.x

Syntax

Timestamp

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Reference

Contains a list of objects you are referenced by.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_HIDDEN_ATTR
DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SF_PER_REPLICA

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Your applications cannot access to this attribute since it is a hidden attribute.

This attribute is kept only on the replica in which it originated. It is not skulked or updated to other replicas. It is used to make it easier to find the objects that have a reference to a certain object. If objects A, B, C, and D all have privileges granted to object X (through use of the ACL attribute), the *Reference* attribute of object X will contain the object names of A, B, C, and D.

This attribute is also used to list bindery object relations in Bindery Services.

Replica

Identifies the name servers that store replicas of a partition.
NetWare Versions: 4.x

Syntax

Replica Pointer

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_READ_ONLY_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Partition

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute identifies a set of name servers that store replicas of the partition. This attribute is multivalued. It is present and nonNULL in every partition entry and may not be directly modified. The set is updated as a side-effect of the operations that create, destroy, or change the replication of a partition.

Replica Up To

Contains a timestamp of what changes the object has sent out to other replicas.
NetWare Versions: : 4.1

Syntax

Octet String

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_READ_ONLY_ATTR
DS_SYNC_IMMEDIATE

Used In

Partition

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Resource (Attribute)

Contains a list of resources associated with the object.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute lists resources that are managed by the server.

Revision

Specifies the revision of the object.
NetWare Versions: 4.x

Syntax

Counter

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Top

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Role Occupant

Specifies the name of an object that fulfills an organizational role.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organizational Role

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

S (State or Province)

Specifies a state or province.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..128)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

External Entity
Locality
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

When used as a component of a Directory name, a state or province name identifies a geographical subdivision in which the named object, is physically located, or with which it is associated in some other important way.

SA (Street Address)

Specifies a site for local distribution and physical delivery in the form of a postal address (that is, the street name, place, avenue, and house number).

NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..128)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

External Entity
Locality
Organization
Organizational Person
Organizational Role
Organizational Unit

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

When used as a component of a Directory name, a street address identifies the address at which the named object is located or with which it is associated in some other important way. The attribute value for street address is a string (for example, "Amulfstrae 60").

SAP Name

Contains the name used by a print server when advertising itself using the NetWare Service Advertising Protocol (SAP).

NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..47)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Print Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Security Equals

Specifies group membership and security equivalences.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SERVER_READ
DS_SYNC_IMMEDIATE
DS_WRITE_MANAGED

Used In

Server (Class)
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Security Flags

Specifies whether the object is using NCP signing.
NetWare Versions: 4.1

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE

Used In

Server (Class)
User (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

See Also

Specifies the names of Directory objects that may reflect other aspects of the same real world object.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Device (Class)
External Entity
Group
List
Locality
Organization
Organizational Role
Organizational Unit
Person
Profile (Class)
Resource (Class)
Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Serial Number

Specifies an identifier that is the serial number of a device.
NetWare Versions: 4.x

Syntax

Printable String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

AFP Server
Device (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Server (Attribute)

Specifies a list of servers.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SERVER_READ
DS_SYNC_IMMEDIATE (4.1)

Used In

Computer
Queue (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Server Holds

Contains the number of accounting charges pending while a server performs a chargeable action.

NetWare Versions: 4.x

Syntax

Hold

Constraints

DS_NONREMOVABLE_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

The *Server Holds* attribute is used when NetWare Accounting is active. Each time a server is about to perform an action that will be charged against a user's account the server makes sure the account has a sufficient balance. The server places the correct number of charges on hold while it performs the function, then deducts those charges from the account balance. While the charges are being held, this attribute value contains user identification and the number of charges that are on hold.

Status

Specifies the operational state of the specified object.
NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Computer
Printer (Class)
Server (Class)
Volume (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Supported Connections

Identifies the number of concurrent connections a server allows.
NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

AFP Server

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Supported Gateway

Contains a list of messaging gateways.
NetWare Versions: 4.1

Syntax

Case Ignore String

Constraints

DS_NONREMOVEABLE_ATTR
DS_SIZED_ATTR (1..4096)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Messaging Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Supported gateways provide messaging connectivity between the MHS messaging system and external messaging systems.

The gateway names consist of two sub-attributes: gateway name and gateway protocol type. The *Supported Gateway* names have the convention of eight or fewer characters followed by "/" followed by another eight or fewer characters.

This attribute is used to support existing third-party gateway products.

Supported Services

Contains a list of services supported by the associated object.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Messaging Server (Class)
NCP Server

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Supported Typefaces

Identifies the typefaces supported by a printer. Multiple typefaces should be represented as multiple values.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Printer (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

An example of a value for this attribute is:

```
Supported Typefaces=computer modern, American modern, CM, AM
```

Surname

Specifies the denotative construct an individual inherits from a parent (or assumes by marriage) and by which the individual is commonly known.

NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_SINGLE_VALUED_ATTR (4.1)
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Person

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

An attribute value for this attribute is a string (for example, "Smith").

Synchronized Up To

A list of time stamps that indicate the last time all servers holding a copy of the specified replica were synchronized.

NetWare Versions: 4.x

Syntax

Timestamp

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SF_PER_REPLICA

Used In

Partition

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Telephone Number (Attribute)

Specifies a telephone number associated with an object.
NetWare Versions: 4.x

Syntax

Telephone Number (Syntax)

Constraints

DS_NONREMOVABLE_ATTR
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Organization
Organizational Role
Organizational Unit
Person

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

An attribute value for this attribute is a string that complies with the internationally agreed format for showing international telephone numbers, Recommendation E.123, (for example, "+ 44 582 10101").

Title

Specifies the designated position or function of an object within an organization.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

External Entity
Organizational Person

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

An attribute value for this attribute is a string. For example:

```
"Manager, Distributed Applications."
```

UID (User ID)

Specifies a unique user ID for use by UNIX clients.
NetWare Versions: 4.x

Syntax

Integer

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

User (Class)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

Unknown (Attribute)

Contains an attribute value for an attribute that is no longer defined by the Schema.

NetWare Versions: 4.x

Syntax

Unknown (Syntax)

Constraints

DS_NONREMOVABLE_ATTR

Used In

(Special)

Remarks

For help in understanding the attribute definition template, see Reading NDS Attribute Type Definitions.

This attribute is provided for use by the server in restoring objects. If an attribute definition has been deleted and an object that is being restored has a value of the deleted attribute type, the now undefined attribute will be restored as *Unknown*.

Unknown Base Class

Stores the class an object was before it was changed to an unknown class.
NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_READ_ONLY_ATTR
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..32)
DS_STRING_ATTR

Used In

(none)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

User (Attribute)

Contains a list of users associated with the object.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SERVER_READ
DS_SYNC_IMMEDIATE (4.1)

Used In

Queue (Class)
Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

This attribute contains a list specifying which users are authorized to use the device, resource, or server.

Version

A string that describes the version identifier of the software associated with the object.

NetWare Versions: 4.x

Syntax

Case Ignore String

Constraints

DS_NONREMOVABLE_ATTR
DS_PUBLIC_READ
DS_SINGLE_VALUED_ATTR
DS_SIZED_ATTR (1..64)
DS_STRING_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Server (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

Volume (Attribute)

Contains the distinguished name of a volume on which a queue is located.
NetWare Versions: 4.x

Syntax

Distinguished Name

Constraints

DS_NONREMOVABLE_ATTR
DS_SINGLE_VALUED_ATTR
DS_SYNC_IMMEDIATE (4.1)

Used In

Queue (Class)

Remarks

For help in understanding the attribute definition template, seeReading NDS Attribute Type Definitions.

NDS Attribute Syntax Definitions

Back Link (Syntax)

Used for the Back Link attribute, which the Directory uses for its internal management

Syntax ID

```
#define SYN_BACK_LINK 23
```

API Data Structure

```
typedef struct
{
   NWOBJ_ID   remoteID;
   pnchar     objectName;
} Back_Link_T;
```

Matching Rules

Equality

Used In

Back Link (Attribute)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Boolean

Used for attributes whose values represent true (1) or false (0)

Syntax ID

```
#define  SYN_BOOLEAN  7
```

API Data Structure

```
typedef nuint8 Boolean_T;
```

Matching Rules

Equality

Used In

Allow Unlimited Credit
Detect Intruder
Locked By Intruder
Lockout After Detection
Login Disabled
Password Allow Change
Password Required
Password Unique Required

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Two boolean attributes match for equality if they are both TRUE or both FALSE.

Any attribute defined using this syntax is single-valued.

Case Exact String

Used in attributes whose values are strings for which the case (upper or lower) is significant when performing comparisons

Syntax ID

```
#define  SYN_CE_STRING  2
```

API Data Structure

```
typedef pnchar CE_String_T;
```

Matching Rules

Equality
Substrings

Used In

None

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Two case exact strings match for equality when they are of same length and their corresponding characters are identical. For example, "Dundee" and "DUNDEE" do not match.

In comparing case exact strings, the following spaces are not significant:

Leading spaces (precede the first printing character)

Trailing spaces (follow the last printing character)

Multiple consecutive internal spaces (equivalent to a single space character)

Attributes conforming to this syntax are matched in a manner omitting those spaces that are not significant (as defined above). There is no guarantee that insignificant spaces will be preserved by the Directory. A name server is free to omit insignificant spaces when storing an attribute value.

Case Ignore List

Used for attributes whose values are ordered sequences of strings for which the case (upper or lower) is not significant when performing comparisons

Syntax ID

```
#define  SYN_CI_LIST  6
```

API Data Structure

```
typedef struct _ci_list
{
   struct _ci_list   *next;
   pnchar            s;
} CI_List_T;
```

Matching Rules

Equality

Used In

Language

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Two case ignore lists match for equality if and only if the number of strings in each is the same, and all corresponding strings match. For two corresponding strings in the list to match they must be the same length and their corresponding characters must be identical (according to the rules for case ignore strings).

When comparing the strings in case ignore lists, the following spaces are regarded as not significant:

Leading spaces (precede the first printing character)

Trailing spaces (follow the last printing character)

Multiple consecutive internal spaces (equivalent to a single space character).

Attributes conforming to this syntax are matched in a manner omitting those spaces that are not significant (as defined above). There is no guarantee that insignificant spaces will be preserved by the Directory. A name server is free to omit insignificant spaces when storing an

A name server is free to omit insignificant spaces when storing an attribute value.

The **NWDSGetAttrVal** function places successive CI_LIST elements in consecutive memory locations. The *value* parameter of the **NWDSGetAttrVal** function should point to enough memory to contain both the NULL-terminated strings along with a CI_LIST structure per list element. (The required length of value can be determined by calling the **NWDSComputeValueSize** function.)

Case Ignore String

Used in attributes whose values are strings for which the case (upper or lower) is not significant when performing comparison

Syntax ID

```
#define  SYN_CI_STRING  3
```

API Data Structure

```
typedef pnchar CI_String_T;
```

Matching Rules

Equality
Substrings

Used In

C (Country)
Cartridge
CN (Common Name)
Description
Full Name
Generational Qualifier
Given Name
Host Resource Name
Initials
L (Locality)
Mailbox ID
Messaging Server Type
NNS Domain
O (Organization)
OU (Organizational Unit)
Physical Delivery Office Name
Postal Code
Postal Office Box
Queue Directory
S (State or Province)
SA (Street Address)
SAP Name
Supported Gateway
Supported Services
Supported Typefaces

Surname
Title
Unknown Base Class
Version

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Two case ignore strings match for equality when they are of the same length and their corresponding characters are identical in all respects except that of case. For example, as case ignore strings, "Dundee" and "DUNDEE" would be equal.

When comparing case ignore strings, the following spaces are not significant:

Leading spaces (precede the first printing character)

Trailing spaces (follow the last printing character)

Multiple consecutive internal spaces (equivalent to a single space character)

Attributes conforming to this syntax are matched in a manner omitting those spaces that are not significant (as defined above). There is no guarantee that insignificant spaces will be preserved by the Directory. A name server is free to omit insignificant spaces when storing an attribute value.

Class Name

Used for attributes whose values represent object class names

Syntax ID

```
#define  SYN_CLASS_NAME  20
```

API Data Structure

```
typedef pnchar Class_Name_T
```

Matching Rules

Equality

Used In

Object Class

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

The matching rule for values of Class Name are the same as those for Case Ignore String.

Counter

Used for attributes whose values are signed integers

Syntax ID

```
#define  SYN_COUNTER  22
```

API Data Structure

```
typedef  nuint32  Counter_T;
```

Matching Rules

Equality
Ordering

Used In

Account Balance
Login Grace Remaining
Login Intruder Attempts
Revision

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Any attribute defined using this syntax is single-valued. This syntax differs from Integer in that any value added to an attribute of this syntax is added to the total, and any value deleted is subtracted from the total.

Distinguished Name

Used for attributes whose values are the names of objects in the Directory tree

Syntax ID

```
#define  SYN_DIST_NAME  1
```

API Data Structure

```
typedef pnchar DN_T;
```

Matching Rules

Equality

Used In

Aliased Object Name
Default Queue
Device (Attribute)
Equivalent To Me
Group Membership
Higher Privileges
Host Device
Host Server
Mailbox Location
Member
Message Routing Group (Attribute)
Message Server
Messaging Server (Attribute)
Operator
Owner
Postmaster
Profile (Attribute)
Profile Membership
Reference
Resource (Attribute)
Role Occupant
Security Equals
See Also
Server (Attribute)
User (Attribute)
Volume (Attribute)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

EMail Address (Syntax)

Used for attributes whose values represent Email addresses

Syntax ID

```
#define  SYN_EMAIL_ADDRESS  14
```

API Data Structure

```
typedef struct
{
   NWEMAIL_TYPE   type;
   pnchar         address;
} EMail_Address_T;
```

Matching Rules

Equality

Used In

EMail Address (Attribute)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Facsimile Telephone Number (Syntax)

Specifies a string that complies with the internationally agreed format for showing international telephone numbers, E.123, and an optional bit string formatted according to Recommendation T.30

Syntax ID

```
#define  SYN_FAX_NUMBER  11
```

API Data Structure

```
typedef struct
{
   pnchar         telephoneNumber;
   Bit_String_T   parameters;
} Fax_Number_T;
```

Matching Rules

Equality

Used In

Facsimile Telephone Number (Attribute)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Facsimile Telephone Number values are matched based on the telephone number field. The rules for matching fax telephone numbers are identical to those for the Case Exact syntax except that all space and hyphen (-) characters are skipped during the comparison.

Hold

Used for attributes whose values represent an object name/level pair

Syntax ID

```
#define  SYN_HOLD  26
```

API Data Structure

```
typedef struct
{
   pnchar          objectName;
   NWHOLD_AMOUNT   amount;
} Hold_T;
```

Matching Rules

Equality

Used In

Server Holds

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Integer

Used for attributes whose values are signed integers

Syntax ID

```
#define  SYN_INTEGER  8
```

API Data Structure

```
typedef nint32 Integer_T;
```

Matching Rules

Equality
Ordering

Used In

Bindery Object Restriction
Convergence
DS Revision
GID (Group ID)
Login Grace Limit
Login Intruder Limit
Login Maximum Simultaneous
Memory
Minimum Account Balance
Password Minimum Length
Security Flags
Status
Supported Connections
UID (User ID)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

The attributes for two integers match for equality if they are the same. The ordering rules for integers apply.

Interval

Used for attributes whose values represent intervals of time in seconds

Syntax ID

```
#define  SYN_INTERVAL  27
```

API Data Structure

```
typedef nint32  Integer_T;
```

Matching Rules

Equality
Ordering

Used In

Certificate Validity Interval
High Convergence Sync Interval
Intruder Attempt Reset Interval
Intruder Lockout Reset Interval
Low Convergence Sync Interval
Password Expiration Interval

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Net Address

Used for network addresses in the NetWare environment. The address type indicates the type of communications protocol used (IPX™, AppleTalk*, etc.)

Syntax ID

```
#define  SYN_NET_ADDRESS  12
```

API Data Structure

```
typedef struct
{
   NWNET_ADDR_TYPE  addressType;
   NWNET_ADDR_LEN   addressLength;
   NWNET_ADDR       *address;
} Net_Address_T;
```

Matching Rules

Equality

Used In

Login Intruder Address
Network Address
Network Address Restriction

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

For two values of Net Address to match, the type, length, and value of the address must match. The address length is the number of bytes. The address itself is stored as a binary string. This string is the literal value of the address. To display it as a hexadecimal value, you must convert each 4-bit nibble to the correct character (0,1,2,3,...F).

Numeric String

Used for attributes whose values are numeric strings as defined in CCITT X.208

Syntax ID

```
#define  SYN_NU_STRING  5
```

API Data Structure

```
typedef pnchar NU_String_T;
```

Matching Rules

Equality
Substrings

Used In

Bindery Type

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

For two numeric strings to match for equality, the strings must be the same length and their corresponding characters must be identical. The following characters are in the numeric string character set:

0..9 digits

Space character

When comparing numeric strings, the following spaces are not significant:

Leading spaces (precede the first printing character)

Trailing spaces (follow the last printing character)

Multiple consecutive internal spaces (equivalent to a single space character)

Attributes conforming to this syntax are matched in a manner omitting those spaces that are not significant (as defined above). Also, there is no guarantee that insignificant spaces will be preserved by the Directory. A name server is free to omit insignificant spaces when storing an attribute value.

Object ACL

Used for attributes whose values represent ACL entries

Syntax ID

```
#define  SYN_OBJECT_ACL  17
```

API Data Structure

```
typedef struct
{
   pnchar            protectedAttrName;
   pnchar            subjectName;
   NWDS_PRIVILEGES   privileges;
} Object_ACL_T;
```

Matching Rules

Approximate Matching
Equality

Used In

ACL
Inherited ACL

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

An Object ACL value can protect either an object or an attribute. The protected object is always the one that contains the ACL attribute. If an ACL entry is to apply to the object as a whole, the protected attribute name should be left empty (NULL). If a specific attribute is to be protected, it should be named in the ACL entry.

An ACL value can be matched against either a subject or a privilege set, or both. If the subject name is not to be considered in the comparison, it should be specified as NULL. If the privilege set is not to be considered in the comparison, an approximate match with a privilege set value of zero should be specified.

The Object ACL syntax supports both matching for equality and approximate matching. The difference between matching for equality and approximate matching concerns the privileges field of the comparison value. When matching for equality, the privilege set must match exactly for the comparison to succeed. When approximate matching has been selected, any bits in the privilege field in the filter that are set must also be set in the target. Any other bits in the target are ignored.

Values with the same *protectedAttrName* and *subjectName* fields are considered to be duplicate, and so are not permitted.

Octet List

Used to describe an ordered sequence of octet strings

Syntax ID

```
#define  SYN_OCTET_LIST  13
```

API Data Structure

```
typedef struct _OCTET_LIST
{
   struct _Octet_List   *next;
   NWLEN                length;
   pnuint8              data;
} Octet_List_T;
```

Matching Rules

Approximate Equals
Equality

Used In

None

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

A presented octet list matches a stored list if the presented list is a subset of the stored list. Octet strings are so designated because they are not interpreted by the Directory. They are simply a series of bits with no Unicode* implications. The *length* is the number of bits divided by 8. Thus each octet represents eight bits of data. The number of data bits will always be evenly divisible by 8.

Octet String

Used for attributes whose values are byte strings uninterpreted by the Directory.

Syntax ID

```
#define  SYN_OCTET_STRING 9
```

API Data Structure

```
typedef struct
{
   NWLEN    length;
   pnuint8  data;
} Octet_String_T;
```

Matching Rules

Equality
Ordering

Used In

Authority Revocation
Bindery Property
CA Private Key
CA Public Key
Certificate Revocation
Cross Certificate Pair
External Name
External Synchronizer
Login Allowed Time Map
Obituary
Passwords Used
Printer Configuration
Private Key
Public Key
Replica Up To

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

For two octet strings to match, they must be the same length and the corresponding bit sequence (octets) must be identical. When comparing two strings, the first pair of octets that do not match are used to determine the order of the strings. These are not Unicode strings.

Path (Syntax)

The Path syntax is used for attributes that represent a file system path

Syntax ID

```
#define  SYN_PATH  15
```

API Data Structure

```
typedef struct
{
   NWNAME_SPACE_TYPE   nameSpaceType;
   pnchar              volumeName;
   pnchar              path;
} Path_T;
```

Matching Rules

Equality

Used In

Home Directory
Messaging Database Location
Path (Attribute)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

The string represented by the *path* field is compared for equality using the same rules that Case Exact String uses.

The *volumeName* field must refer to a volume object that already exists in the Directory.

Postal Address (Syntax)

Used for attributes whose values are postal addresses

Syntax ID

```
#define  SYN_PO_ADDRESS  18
```

API Data Structure

```
typedef pnchar Postal_Address_T[6];
```

Matching Rules

Equality

Used In

Postal Address (Attribute)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

An attribute value for Postal Address will be typically composed of selected attributes from the MHS Unformatted Postal O/R Address version 1 according to Recommendation F.401. The value is limited to 6 lines of 30 characters each, including a Postal Country Name. Normally the information contained in such an address could include a name, street address, city, state or province, postal code and possibly a postal office box number depending on the specific requirements of the named object.

The matching rules for values of this type are the same as those for Case Ignore List.

Printable String

Used in attributes whose values are printable strings as defined in CCITT X.208. The case (upper or lower) is significant when comparing printable strings

Syntax ID

```
#define  SYN_PR_STRING  4
```

API Data Structure

```
typedef pnchar PR_String_T;
```

Matching Rules

Equality
Substrings

Used In

Page Description Language
Serial Number

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

The following characters are in the printable string character set:

A.. Z	upper case alphabetic characters
a..z	lower case alphabetic characters
0..9	digits
	space character
'	apostrophe
(	left parenthesis
)	right parenthesis
+	plus sign
,	comma
-	hyphen
.	full stop (period)

/	solidus (slash)
:	colon
=	equal sign
?	question mark

Two printable strings match for equality when they are the same length and their corresponding characters are identical. For example, as printable strings, "Dundee" and "DUNDEE" do not match.

When comparing printable strings, the following spaces are not significant:

Leading spaces (precede the first printing character)

Trailing spaces (follow the last printing character)

Multiple consecutive internal spaces (equivalent to a single space character)

Attributes conforming to this syntax are matched in a manner omitting those spaces that are not significant (as defined above). There is no guarantee that insignificant spaces will be preserved by the Directory. A name server is free to omit insignificant spaces when storing an attribute value.

Replica Pointer

Used for attributes whose values represent partition replicas

Syntax ID

```
#define  SYN_REPLICA_POINTER   16
```

API Data Structure

```
typedef struct
{
   pnchar           serverName;
   NWREPLICA_TYPE   replicaType;
   NWREPLICA_NUM    replicaNumber;
   NWCOUNT          count;
   Net_Address_T    replicaAddressHint[?];
} Replica_Pointer_T;
```

Matching Rules

Equality

Used In

Replica

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Each value of this syntax is composed of five parts:

The complete name of the name server that stores the replica

A value describing the capabilities of this copy of the partition: master, secondary, read-only

A number representing the replica

A value indicating the number of replicas that exist

A network address giving a hint for a node at which the name server probably resides.

The matching rules for values of Replica Pointer are based on the replica server name field only, and are the same as those for Distinguished Name.

The length of the *replicaAddressHint* structure is variable and can be calculated by multiplying the *count* field by the length of the Net_Address_T structure (that is, count multiplied by 9).

Stream

Used for login scripts and other stream attributes

Syntax ID

```
#define  SYN_STREAM  21
```

API Data Structure

```
typedef struct
{
   NWLEN     length;(always 0)
   pnuint8   data;
} Stream_T;
```

Matching Rules

None

Used In

Login Script
Print Job Configuration
Printer Control

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

Any attribute defined with this syntax is single-valued. When returned from read or search, it returns an empty octet string. When added, the value is ignored. The value of any attribute of this syntax must be read or written by calling the **NWDSOpenStream** function, followed by calling standard file Read/Write functions.

Telephone Number (Syntax)

Used for attributes whose values are telephone numbers

Syntax ID

```
#define  SYN_TEL_NUMBER  10
```

API Data Structure

```
typedef pnchar TN_String_T;
```

Matching Rules

Equality
Substrings

Used In

Telephone Number (Attribute)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

The length of telephone number strings must be between 1 and 32. The rules for matching telephone numbers are identical to those for the Case Exact attribute syntax except that all space and hyphen (-) characters are skipped during the comparison.

Time

Used for attributes whose values represent time

Syntax ID

```
#define  SYN_TIME  24
```

API Data Structure

```
typedef nint32  Integer_T;
```

Matching Rules

Equality
Ordering

Used In

Last Login Time
Login Expiration Time
Login Intruder Reset Time
Login Time
Low Convergence Reset Time
Password Expiration Time

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

A Time value consists of a whole number of seconds, where zero equals 12:00 midnight, January 1, 1970, UTC.

Two Time values are compared by comparing the Integer_T values.

Timestamp

Used for attributes whose values mark the time when a particular event occurred or will occur

Syntax ID

```
#define  SYN_TIMESTAMP  19
```

API Data Structure

```
typedef struct
{
   NWSECONDS     wholeSeconds;
   NWDS_EVENT    eventID;
} NWDS_TimeStamp_T;
```

Matching Rules

Equality
Ordering

Used In

Replica Up To
Partition Creation Time
Received Up To
Synchronized Up To

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

A Timestamp value consists of a whole number of seconds, where zero equals 12:00 midnight, January 1, 1970, UTC. The event ID is an integer which further orders events occurring within the same whole-second interval.

Two Time values are compaired by comparing the whole second fields and then the event ID fields. If the whole seconds fields are unequal, order is determined by that field alone. If the whole seconds fields are equal and the event ID fields are unequal, order is determined by the event ID fields. If both fields are equal, the time stamps are equal.

Typed Name

Used for attributes whose values represent a level and an interval associated with an object

Syntax ID

```
#define  SYN_TYPED_NAME  25
```

API Data Structure

```
typedef struct
{
   pnchar             objectName;
   NWDS_TYPE_LEVEL    level;
   NWDS_INTERVAL      interval;
} Typed_Name_T;
```

Matching Rules

Equality

Used In

Notify
Partition Control
Print Server (Attribute)
Printer (Attribute)
Queue (Attribute)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.

The level of the attribute indicates the priority. The interval indicates the frequency of reference.

The values are user-assigned and relative. To be effective they must be implemented by the user. The user can use them to implement iterative intervals or to enforce priorities.

Unknown (Syntax)

Used for attributes whose attribute definition was deleted from the schema

Syntax ID

```
#define  SYN_UNKNOWN  0
```

API Data Structure

```
typedef struct
{
   pnchar        attrName;
   NWSYNTAX_ID   syntaxID;
   NWLEN         valueLen;
   void          *value;
} Unknown_Attr_T;
```

Matching Rules

None

Used In

Unknown (Attribute)

Remarks

For help in understanding the syntax definition template, see Reading Syntax Definitions.