Chapter 2

Attribute Syntax Definitions

Overview

This chapter describes the attribute syntax definitions or primary data types for values stored in the Directory. An attribute syntax consists of a single data type for which syntax matching rules and qualifiers or directives have been specified.

Attribute types are built on attribute syntaxes. The specifications for the standard set of attribute syntaxes used to create Directory attribute types are presented here. You can create new attribute types using these syntaxes, but you cannot create any new syntax definitions. The syntax definitions are static definitions represented in basic C code format.

The attribute syntaxes are presented in alphabetical order for easy access. You should read this chapter if you wish to learn the specific details about the attribute syntaxes and how they function in the Directory.

Reading the Syntax Definitions

For each syntax specification, the name of the syntax appears in large type at the first of the listing. This is followed by a brief description of how the syntax is used. Each syntax is defined in terms of a syntax ID, data formats, matching rules, and the attribute types it is used in. A remarks section may be included at the end giving additional information concerning comparisons, explanation of structure members, syntax examples, etc.

The notation used to define the attribute syntaxes is defined below.

Syntax ID

The *Syntax ID* is a 32-bit integer used to identify attribute values. The 28 *Syntax IDs* are defined in the DCONST.H file for internal programming or NLM application development.

Data Formats

Two different data formats are defined for each attribute syntax.

- The “Transfer Syntax” defines the specific transmission encoding used in NDS™ request/reply data packets. When an attribute’s value is transmitted on the wire, it conforms to the attribute’s syntax encoding.
- The “API Data Structure” is a C code structure supported by the Novell Directory Services™ API. The data structure for each syntax and the syntax IDs are defined in the file NWDSDEFS.H for client applications.

Syntax Matching Rules

Matching rules indicate what characteristics are significant when comparing two values of the same syntax. The three primary matching rules are *equality*, *substrings*, and *ordering*. Any one or more of these matching rules can apply to an attribute syntax.

To match for *equality*, two equal values must conform to the attribute syntax data type. The Directory does not attempt to match two syntax values that do not specify a match for equality. However, most syntaxes do specify a match for equality.

The *substring* matching rule involves comparing two sub-strings. String syntaxes can include the asterisk (*) wildcard in the pattern to be matched.

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

Two syntaxes use variations of the matching rules: The “Object ACL” syntax uses an *Approximate Matching* rule as well as the *equality* matching rule, and the “Octet List” syntax uses an *Approximate Equals* rule as well as the *equality* matching rule. These two syntax variations are syntax dependent and specific in their usage. The remarks section for each of these two syntaxes discusses these matching rule variants in more detail.

Used In

The attribute type definitions that use the syntax are listed here. To determine which syntax definitions are used for a particular attribute type, refer to the attribute’s syntax specification in Chapter 3, “Attribute Type Definitions.”

Individual Attribute Syntax Definitions

This section contains the detailed definition for each of the 28 syntaxes used by NDS release 611 attributes.

Back Link

Description

The *Back Link* attribute uses this syntax to keep track of other servers referring to an NDS object. The Directory uses the *Back Link* attribute for internal management.

Syntax ID

#define SYN_BACK_LINK 23

Data Formats

Transfer Syntax

```
uint32      Length;
unicode     Distinguished Name;
uint32      EntryID;
```

API Data Structure

```
typedef struct
{
    char        *remoteID;
    uint32      *objectName;
}   Back_Lint_T;
```

Matching Rules

Equality

Used In

Back Link

Remarks

In the transfer syntax, the *Distinguished Name* (unicode) field identifies the server holding a reference, and the *EntryID* (uint32) field identifies the reference that is valid on the server. In the API data structure, the remoteID field points to the *DistinguishedName* and the *ObjectName* points to the *EntryID*.

Boolean

Description

This syntax is used by attributes whose values represent *True* and *False*.

Syntax ID

#define SYN_BOOLEAN 7

Data Formats

Transfer Syntax

uint32	Length = 1
BYTE	Content

API Data Structure

typedef uint8 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

In the *Boolean* syntax, *True* is represented as one (1), while *False* is represented as zero (0). 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

Description

This syntax is used by attributes whose values are Unicode strings that are case-sensitive in comparison operations.

Syntax ID

#define SYN_CE_STRING 2

Data Formats

Transfer Syntax

uint32	Length
unicode	String

API Data Structure

typedef char *CE_String_T;

Matching Rules

Equality
Substring

Used In

None

Remarks

Two *Case Exact Strings* match for equality when they are of the same length and their corresponding characters, including case, are identical.

In comparing *Case Exact Strings*, the following white space (spaces, tabs, etc.) is not significant:

- Leading spaces (those preceding the first printable character)
- Trailing spaces (those following the last printable character)
- Multiple consecutive internal spaces (these are taken as equivalent to a single space character)

In matching attributes that conform to this syntax, NDS omits those spaces that are not significant (as defined above). The Directory may omit insignificant spaces when storing such an attribute value.

Case Ignore List

Description

This syntax is used by attributes whose values are ordered sequences of Unicode strings that are case insensitive in comparisons operations.

Syntax ID

#define SYN_CI_LIST 6

Data Formats

Transfer Syntax

uint32	Length of attribute value
uint32	No. of strings = M
unicode [M]	String

API Data Structure

```
typedef  struct  _ci_list
{
     struct  _ci_list     *next;
     char                 *s;
}    CI_List_T;
```

Matching Rules

Equality

Used In

Language

Remarks

Two *Case Ignore Lists* match for equality if the number of strings in each is the same and all corresponding strings match. For two corresponding strings in the list to match, their lengths and corresponding characters must be identical (according to the matching rules for the *Case Ignore String* syntax).

When comparing the strings in attributes using the *Case Ignore List* syntax, the following white space (spaces, tabs, etc.) is regarded as not significant:

- Leading spaces (those preceding the first printable character)
- Trailing spaces (those following the last printable character)
- Multiple consecutive internal spaces (these are taken as equivalent to a single space character)

In matching attributes that conform to this syntax, NDS omits those spaces that are not significant (as defined above). The Directory may omit insignificant spaces when storing such an attribute value.

The *NWDSGetAttrVal* function places successive CI-LIST elements in consecutive memory locations. The value parameter of *NWDSGetAttrVal* 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 this value can be determined by calling *NWDSComputeAttrValSize*.

Case Ignore String

Description

This syntax defines values which are Unicode strings and case insensitive in comparison operations.

Syntax ID

#define SYN_CI_STRING 3

Data Formats

Transfer Syntax

uint32 Length
unicode String

API Data Structure

typedef char *CI_String_T;

Matching Rules

Equality
Substring

Used In

Cartridge
CN (Common Name)
C (Country Name)
Description
Full Name
Generational Qualifier
Given Name
Host Resource Name
Initials
L (Locality Name)
Mailbox ID
Messaging Server Type
NNS Domain
O (Organization Name)
OU (Organizational Unit Name)
Physical Delivery Office Name
Postal Code
Postal Office Box
Queue Directory
SAP Name
S (State or Province Name)
SA (Street Address)
Supported Gateway
Supported Services
Supported Typefaces
Surname
T (Tree Name)
Title
Unknown Base Class
Version

Remarks

Two *Case Ignore Strings* match for equality when their lengths and corresponding characters are identical in all respects except that of case and spaces as described below. For example, as *Case Ignore Strings*, “Dundee” and “DUNDEE” would be equal.

In comparing attribute strings that use the *Case Ignore String* syntax, the following white space (spaces, tabs, etc.) is not significant:

- Leading spaces (those preceding the first printable character)

- Trailing spaces (those following the last printable character)
- Multiple consecutive internal spaces (these are taken as equivalent to a single space character)

In matching attributes that conform to this syntax, NDS omits those spaces that are not significant (as defined above). The Directory may omit insignificant spaces when storing such an attribute value.

Class Name

Description

This syntax specifies values which are NDS object class names.

Syntax ID

#define SYN_CLASS_NAME 20

Data Formats

Transfer Syntax

uint32	Length
unicode	String

API Data Structure

```
typedef char  *Class_Name_T;
```

Matching Rules

Equality

Used In

Object Class

Remarks

The matching rule for values of *Class Name* are the same as those for *Case Ignore String*. That is, two *Class Names* match for equality when their lengths and corresponding characters are identical in all respects except that of case and spaces as described below.

In comparing attributes using the *Class Name* syntax, the following white space (spaces, tabs, etc.) is not significant:

- Leading spaces (those preceding the first printable character)
- Trailing spaces (those following the last printable character)
- Multiple consecutive internal spaces (these are taken as equivalent to a single space character)

In matching attributes that conform to this syntax, NDS omits those spaces that are not significant (as defined above). The Directory may omit insignificant spaces when storing such an attribute value.

Counter

Description

This syntax specifies values which are incrementally-modified, numeric, signed integers.

Syntax ID

#define SYN_COUNTER 22

Data Formats

Transfer Syntax

uint32	Length = 4
uint32	Content

API Data Structure

typedef uint32 Counter_T;

Matching Rules

Equality
Ordering

Used In

Account Balance
Login Grace Remaining
Login Intruder Attempts
Revision

Remarks

Any attribute defined using *Counter* is single-valued. This syntax differs from *Integer* in that any value added to an attribute of this syntax is arithmetically added to the total, and any value deleted is arithmetically subtracted from the total.

Distinguished Name

Description

This syntax defines values which are the names of objects in the Directory tree.

Syntax ID

#define SYN_DIST_NAME 1

Data Formats

Transfer Syntax

uint32	Length
unicode	String

API Data Structure

```
typedef  char  *DN_T;
```

Matching Rules

Equality

Used In

Aliased Object Name
Default Queue
Device
Equivalent To Me
Group Membership
Higher Privileges
Host Device
Host Server
Mailbox Location
Member
Message Routing Group
Message Server
Messaging Server
Operator
Owner
Postmaster
Profile
Profile Membership
Reference
Resource
Role Occupant
Security Equals
See Also
Server
User
Volume

Remarks

A *Distinguished Name* (DN) is transmitted on the wire as a Unicode string. The largest supported DN is 256 Unicode characters, not including the terminating NULL character. *Distinguished Names* are not case sensitive, even if one of the naming attributes is case sensitive.

EMail Address

Description

This syntax defines values which are strings of binary information.

Syntax ID

#define SYN_EMAIL_ADDRESS 14

Data Formats

Transfer Syntax

```
uint32          Length = N
BYTE [N]        Content
```

API Data Structure

```
typedef  struct
{
     uint32          type;
     char            *address;
}    EMail_Address_T;
```

Matching Rules

Equality

Used In

EMail Address

Remarks

On the wire, this syntax representation is the same as an *Octet String*. NDS makes no assumption about the internal structure of the *Content* field.

Only the *EMail Address* attribute uses this syntax.

Facsimile Telephone Number

Description

This syntax defines a string that complies with the format agreed upon for storing international telephone numbers, E.123, and an optional bit string formatted according to Recommendation T.30.

Syntax ID

#define SYN_FAX_NUMBER 11

Data Formats

Transfer Syntax

uint32	Length
unicode	Telephone number
uint32	Bit Count = B
uint32	Bit String Length = M
BYTE (M)	Bit String

API Data Structure

```
typedef  struct
{
    uint32      numOfBits;
    uint8       *data;
}
```

Matching Rules

Equality

Used In

Facsimile Telephone Number

Remarks

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 String* syntax except that all space and hyphen (-) characters are ignored during the comparison.

This syntax combines a telephone number with an optional bit string. On the wire, the bit string is represented using an integral number of bytes, with padding bits (if needed) at the least-significant end of the final byte. The value *M* is the number of bytes needed to hold *B* bits, where the value *B* indicates the number of bits in the bit string. That is, the value of *M* is *B*/8 rounded up to the nearest integer.

Hold

Description

This syntax defines values which are accounting quantities, whose values are signed integers.

Syntax ID

#define SYN_HOLD 26

Data Formats

Transfer Syntax

```
uint32      Length
uint32      Amount
unicode     Subject
```

API Data Structure

```
typedef  struct
{
    char        *objectName
    uint32      amount;
}   Hold_T;
```

Matching Rules

Equality

Used In

Server Holds

Remarks

This syntax is an accounting quantity, which is an amount tentatively held against a subject's credit limit, pending completion of a transaction. In the transfer syntax, the *Subject* field is the *Distinguished Name* of the object. NDS treats the *Hold* amount similarly to the *Counter* syntax, with new values added to or subtracted from the base total. If the evaluated *Hold* amount goes to 0 (zero), the *Hold* record is deleted.

Integer

Description

This syntax is used by attributes represented as signed numeric values.

Syntax ID

#define SYN_INTEGER 8

Data Formats

Transfer Syntax

uint32	Length = 4
uint32	Content

API Data Structure

typedef uint32 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

Two *Integer* value attributes match for equality if they are identical. The comparison for ordering uses signed integer rules.

Interval

Description

This syntax is used by attributes whose values are signed numeric integers and represent intervals of time.

Syntax ID

#define SYN_INTERVAL 27

Data Formats

Transfer Syntax

uint32	Length = 4
uint32	Content

API Data Structure

typedef uint32 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

The *Interval* syntax uses the same representation as the *Integer*. The *Interval* value is the number of seconds in a time interval.

Net Address

Description

This syntax represents a network-layer address in the NetWare environment.

Syntax ID

#define SYN_NET_ADDRESS 12

Data Formats

Transfer Syntax

```
uint32          Length
uint32          Transport Type
uint32          Address Length = M
BYTE [M]        Address
```

API Data Structure

```
typedef  struct
{
    uint32      addressType;
    uint32      address Length;
    uint8       *address;
}   Net_Address_T;
```

Matching Rules

Equality

Used In

Login Intruder Address
Network Address
Network Address Restriction

Remarks

The address or transport type indicates the type of communications protocol used: 0 = IPX, 1 = IP, 4 = OSI NSAP, and 5 = AppleTalk. For IPX, the *Address Length* is 12. It includes the network number, host number, and socket, in that order. The address is in binary (not displayable) format.

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, convert each 4-bit nibble to the correct character (0, 1, 2, 3, . . . F).

Numeric String

Description

This syntax is used by attributes whose values are numeric strings as defined in the CCITT X.208 definition of *Numeric String*.

Syntax ID

#define SYN_NU_STRING 5

Data Formats

Transfer Syntax

uint32 Length
unicode String

API Data Structure

typedef char *NU_String_T;

Matching Rules

Equality
Substring

Used In

Bindery Type

Remarks

For two *Numeric Strings* to match for equality, the strings' lengths and 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 white space (spaces, tabs, etc.) is not significant.

- Leading spaces (those preceding the first printable character)
- Trailing spaces (those following the last printable character)
- Multiple consecutive internal spaces (these are taken as equivalent to a single space character).

In matching attributes that conform to this syntax, NDS omits those spaces that are not significant (as defined above). The Directory may omit insignificant spaces when storing such an attribute value.

Object ACL

Description

This syntax is used by attributes whose values represent *Access Control List* (ACL) entries.

Syntax ID

#define SYN_OBJECT_ACL 17

Data Formats

Transfer Syntax

uint32	Length
unicode	Name of Protected Attribute
	Align4
unicode	Subject Name
uint32	Privileges

API Data Structure

```
typedef  struct
{
    char        *protectedAttrName;
    char        *subjectName;
    uint32      privileges;
}   Object_ACL_T;
```

Matching Rules

Approximate Matching
Equality

Used In

ACL
Inherited ACL

Remarks

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, leave the protected attribute name empty (NULL). If a specific attribute is to be protected, name it in the *ACL* entry.

You can match an *ACL* value against either a subject (trustee) or a privilege set, or both. If the subject name is not to be considered in the comparison, specify it as NULL. If the privilege set is not to be considered in the comparison, specify an “approximate match” with a privilege set value of zero.

The *Object ACL* syntax supports both “equality” and “approximate matching” rules. 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. If you have selected “approximate matching,” set any bits in the target that are set in the filter’s privilege field. Any other bits in the target are ignored. Values with the same *protectedAttrName* and *subjectName* are considered to be duplicate and thus are not permitted.

The *Protected Attribute Name* conveys the attribute that the *ACL* is protecting. Three predefined *Protected Attribute Names* have special significance. These names are listed below with an explanation of what

they represent. Note that the brackets are included when the *Protected Attribute Names* and the *Subject Names* appear in a packet on the wire.

[Entry Rights]	Privileges apply to the entire NDS object, rather than an attribute.
[All Attributes Rights]	Privileges apply to all attributes of the object.
[SMS Rights]	Privileges that pertain to the Storage Management System for operations such as backup and restoration.

The *Subject Name* is the *Distinguished Name* of the NDS object being granted a privilege. Certain predefined *Subject Names* have special significance. These are listed below with an explanation of what they represent.

[Self]	Indicates the user authenticated in the current NCP Connections. This can only be used in the *Add Entry* operation.
[Creator]	The user who created the NDS object. This can only be used in the *Add Entry* operation.
[Public]	Includes all objects in the NDS tree.
[Inheritance Mask]	Filters or masks the privileges granted to an object.
[Root]	Denotes the Directory tree root object.

The *Privilege* field is a bit mask that represents the kind of access being granted (value 1) or denied (value 0). The interpretation depends on how the bits are used, as shown in Table 2.1 below.

Table 2.1
Access Conrtol Bit Masks

Bit Mask	Individual Attributes	[Entry Rights]	[SMS Rights]
0x00000001	Compare Attributes	Browse Entry	Scan
0x00000002	Read Attribute	Add Entry	Backup
0x00000004	Write, Add, Delete Attr.	Delete Entry	Restore
0x00000008	Add/Delete Self	Rename Entry	Rename
0x00000010	(none)	Supervisory	Delete
0x00000020	Supervisory	(none)	Administer

Octet List

Description

This syntax describes an ordered sequence of strings of binary information or *Octet Strings*.

Syntax ID

```
#define  SYN_OCTET_LIST 13
```

Data Formats

Transfer Syntax

```
uint32              Length
uint32              Number of Strings = M
Component String[M] Strings

Each Component String has the following representation:
                    Align4
uint32              Component Length = K
BYTE[K]             Component Data
```

API Data Structure

```
typedef  struct octet_list
{
     struct  _octet_List     *next;
     uint32                  length;
     uint8                   *data;
}    Octet_List_T;
```

Matching Rules

Approximate Equals
Equality

Used In

None

Remarks

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 the Directory does not interpret them. They are simply a series of bits with no Unicode implications. When comparing two *Octet Lists*, only one *Octet String* of the lists need match, thus the “Approximate Equals” matching rule applies. The *length* is the number of bits divided by eight and rounded up to the nearest integer (see *Facsimile Telephone Number* remarks). Thus, each octet represents eight bits of data. That is, the number of data bits will always be evenly divisible by eight.

Octet String

Description

This syntax is used by attributes whose values are strings of binary information not interpreted by the Directory.

Syntax ID

#define SYN_OCTET_STRING 9

Data Formats

Transfer Syntax

```
uint32          Length = N
BYTE[N]         Content
```

API Data Structure

```
typedef  struct
{
    uint32      length;
    uint8       *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
Permanent Config Parms
Printer Configuration
Private Key
Public Key
Replica Up To

Remarks

These *Octet Strings* are non-Unicode strings. For two *Octet Strings* to match, their lengths and corresponding bit sequences (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.

Path

Description

This syntax is used by attributes that represent a file system path, containing complete information to locate a file on a NetWare server.

Syntax ID

#define SYN_PATH 15

Data Formats

Transfer Syntax

uint32	Length
uint32	Name Space
unicode	Volume
	Align4
unicode	File Name

API Data Structure

```
typedef  struct
{
     uint32      nameSpaceType;
     char        *volumeName;
     char        *path;
}    Path_T;
```

Matching Rules

Equality

Used In

Home Directory
Messaging Database Location
Path

Remarks

The string represented by *Path* is compared for equality using the same rules as *Case Exact String*s. That is, two *Paths* match for equality when their lengths and corresponding characters, including case, are identical.

In comparing two *Paths*, the following white space (spaces, tabs, etc.) is not significant:

- Leading spaces (those preceding the first printable character)
- Trailing spaces (those following the last printable character)
- Multiple consecutive internal spaces (these are taken as equivalent to a single space character)

In matching attributes that conform to this syntax, NDS omits those spaces that are not significant (as defined above). The Directory may omit insignificant spaces when storing such an attribute value.

The *Volume Name* field is the *Distinguished Name* of the *Volume* object. The *Volume* object must already exists in the Directory. Within the *Volume*, the file's name is indicated by the *File Name* field.

Rules for operating on the *File Name* depend on the *Name Space*, which is defined as follows:

- 0 = DOS
- 1 = Macintosh
- 2 = UNIX
- 3 = FTAM
- 4 = OS/2

Postal Address

Description

This syntax is used by attributes whose values are Unicode strings of *Postal Addresses*.

Syntax ID

#define SYN_PO_ADDRESS 18

Data Formats

Transfer Syntax

uint32	Length
uint32	Number of Strings = M
unicode[M]	Strings

API Data Structure

typedef char *Postal_Address_T[6];

Matching Rules

Equality

Used In

Postal Address

Remarks

An attribute value for *Postal Address* is typically composed of selected attributes from the MHS "Unformatted Postal O/R Address" specification 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 an addressee's 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. A blank line is a zero-length Unicode string.

The matching rules for values of this type are the same as those for *Case Ignore List*. That is, two *Postal Addresses* 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 (see the rules for *Case Ignore String*).

When comparing the strings in a *Postal Address*, the following white space (spaces, tabs, etc.) is regarded as not significant:

- Leading spaces (those preceding the first printable character)
- Trailing spaces (those following the last printable character)
- Multiple consecutive internal spaces (these are taken as equivalent to a single space character)

In matching attributes that conform to this syntax, NDS omits those spaces that are not significant (as defined above). The Directory may omit insignificant spaces when storing such an attribute value.

Printable String

Description

This syntax is used by attributes whose values are printable strings, as defined in CCITT X.208.

Syntax ID

#define SYN_PR_STRING 4

Data Formats

Transfer Syntax

uint32 Length
unicode String

API Data Structure

typedef char *PR_String_T;

Matching Rules

Equality
Substrings

Used In

Page Description Language
Serial Number

Remarks

The printable string character set includes the following characters:

A..Z	uppercase alphabetic characters
a..z	lowercase alphabetic characters
0..9	digits
	space character
‘	apostrophe
( )	left and 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. Case (upper or lower) is significant when comparing printable strings.

When comparing printable strings, the following white space (spaces, tabs, etc.) is insignificant.

- Leading spaces (those preceding the first printable character)
- Trailing spaces (those following the last printable character)
- Multiple consecutive internal spaces (these are taken as equivalent to a single space character)

In matching attributes that conform to this syntax, NDS omits those spaces that are not significant (as defined above). The Directory may omit insignificant spaces when storing such an attribute value.

Replica Pointer

Description This syntax is used by attributes whose values represent partition replicas.

Syntax ID #define SYN_REPLICA_POINTER 16

Data Formats

Transfer Syntax

```
uint32              Length
unicode             Server Name
                    Align4
uint16              Replica Type
uint16              Replica State
uint32              Replica Number
uint32              Number of Addresses = N
Address Record[N]   Addresses
```

API Data Structure

```
typedef  struct
{
    char            *serverName;
    uint32          replicaType;
    uint32          replicaNumber;
    uint32          count;
    NetAddress_T    replicaAddressHint[?];
}   Replica_Pointer_T;
```

Matching Rules Equality

Used In Replica

Remarks Each partition of an NDS tree can have replicas on different servers. A *Replica Pointer* identifies one of these replicas. Each value of this syntax is composed of six parts:

1. The *Server Name* is the *Distinguished Name* of the server that stores the replica.

2. The *Replica Type* is a value describing the capabilities of this copy of the partition replica: 0 = *Master,* 1 = *Secondary,* 2 = *Read-only,* and 3 = *Subordinate Reference*.

3. The *Replica Number* is a number representing this server's copy of the partition. It is unique within the partition (all replicas of a partition have different numbers). This is assigned when the replica is created on a server.

4. The *Number of Addresses* or *replicaAddressHints* is a value indicating the number of address "hints" represented.

5. The network *Address Record* gives a "hint" for a node at which the NDS server probably resides. In transfer syntax, it is the referral containing transport address data. Because a server can be accessible over different protocols, it may have a different address for each pro-

tocol. In the transfer syntax, each *Address Record* has the following representation:

```
Align4
uint32      Transport Type (0=IPX, 1=IP, 4=OSI NSAP,
                5=AppleTalk)
uint32      Address Length = K
BYTE[K]     Address Value
```

6. In the transfer syntax, the *Replica State* is an unsigned integer value representing the progress of a partition operation. In the course of a partition operation, each replica goes through a series of state transitions. The *Replica State* is stored in the most significant 16 bits of the *Replica Type* field of the *Replica Attribute*. The allowable values of a *Replica State* are as follows:
   - 0 = On
   - 1 = New Replica
   - 2 = Dying Replica
   - 3 = Locked
   - 4 = Change Replica Type State 0
   - 5 = Change Replica Type State 1
   - 6 = Transition On
   - 48 = Split State 0
   - 49 = Split State 1
   - 64 = Join State 0
   - 65 = Join State 1
   - 66 = Join State 2
   - 80 = Move State 0
   - 81 = Move State 1
   - 82 = Move State 2

   The matching rules for values of *Replica Pointer* syntaxes are based on the replica *Server Name* field.

   In the API data structure, the *replicaAddressHint* length is variable and cannot be calculated in a block, because address types are not all the same length.The IPX address length is 1; however, other types are different lengths. If only the IPX address is used, calculate the *replicaAddressHint* length by *count* multiplied by the length of *Net_Address_T*, which in this case is 12.

Stream

Description

This syntax represents arbitrary binary information. The *Stream* syntax provides a way to make an NDS attribute out of a file on a file server. Login scripts and other stream attributes use this syntax.

Syntax ID

#define SYN_STREAM 21

Data Formats

Transfer Syntax

uint32	Length = N
BYTE[N]	Content

API Data Structure

typedef Octet_String_T Stream_T;

Matching Rules

None

Used In

Login Script
Print Job Configuration
Printer Control
Type Creator Map

Remarks

Streams are files of information. The data stored in a stream file has no syntax enforcement of any kind. It is purely arbitrary data, defined by the application that created and uses it.

Any attribute defined with this syntax is single-valued. When you try to read an attribute of this type or search on it, the value behaves like an empty string. That is, when you return from read or search, it returns an empty octet string. When you try to add a value, the request is ignored.

You must read or write attributes of this syntax using the *NWDSOpenStream* function followed by normal remote file access operations, including data access (Read/Write functions) and closing the file.

Telephone Number

Description

This syntax is used by attributes whose values are telephone numbers.

Syntax ID

#define SYN_TEL_NUMBER 10

Data Formats

Transfer Syntax

uint32 Length
unicode String

API Data Structure

typedef char *TN_String_T;

Matching Rules

Equality
Substrings

Used In

Telephone Number

Remarks

The length of *Telephone Number* strings must be between 1 and 32 characters. The rules for matching *Telephone Numbers* are identical to those for the *Case Exact String* attribute syntax, except that all spaces and hyphen (-) characters are skipped during comparison.

Two telephone numbers match for equality when their lengths and corresponding characters, including case, are identical.

In the comparisons, the following white space (spaces, tabs, etc.) is not significant:

- Leading spaces (those preceding the first printable character)
- Trailing spaces (those following the last printable character)
- Multiple consecutive internal spaces (these are taken as equivalent to a single space character)

In matching attributes that conform to this syntax, NDS omits those spaces that are not significant (as defined above). The Directory may omit insignificant spaces when storing such an attribute value.

Time

Description

This syntax is used by attributes whose values are unsigned integers and represent time expressed in seconds.

Syntax ID

#define SYN_TIME 24

Data Formats

Transfer Syntax

uint32 Length = 4
uint32 Content

API Data Structure

typedef uint32 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

The *Time* syntax uses the same representation as *Integer*. Comparisons for ordering use unsigned integer rules.

A time value consists of a whole number of seconds since 12:00 midnight, 1 January 1970, UTC.

Timestamp

Description

This syntax is used by attributes whose values mark the time when a particular event occurred.

Syntax ID

#define SYN_TIMESTAMP 19

Data Formats

Transfer Syntax

uint32 Length = 8
TIMESTAMP Timestamp Value
The Timestamp Value is represented as follows:
uint32 Seconds
uint16 Replica Number
uint16 Event

API Data Structure

```
typedef  struct
{
    uint32      wholeSeconds;
    uint16      replicaNum;
    uint16      eventID;
}   TimeStamp_T;
```

Matching Rules

Equality
Ordering

Used In

Last Referenced Time
Partition Creation Time
Received Up To
Synchronized Up To

Remarks

When a significant event occurs, an NDS server mints a new *Time-stamp* value and associates the value with the event. Every *Time-stamp* value is unique within an NDS partition. This provides a total ordering of events occurring on all servers holding replicas of a partition. The representation of a *Timestamp* on the wire as an attribute value is the *Timestamp Value* with a *Length* field of eight-bytes prepended. The *Timestamp Value* has three components:

- The *Seconds* field consists of a whole number of seconds, where zero equals 12:00 midnight,1 January 1970, UTC.
- The *Replica Number* identifies the server that minted the *Time-stamp.* It is assigned whenever a replica is created on a server.
- The *Event* is a sequence number. It is an integer, which further orders events occurring within the same whole-second interval. The *Event* number restarts at one for each new second. The initial NULL value of a *Timestamp* has *Seconds*=1 and *Event*=0. Values can be skipped, but must not be reused. An “unknown” event is coded as 0xFFFF.

Time stamps can be compared for equality and for ordering. Two *Timestamp* values are matched for equality by comparing the whole *Seconds* fields and then the *Event* fields. If the whole *Seconds* fields are

unequal, order is determined by that field alone. If the *Seconds* fields are equal and the *Event* fields are unequal, order is determined by the *Event* fields. If both fields are equal, the time stamps are equal. For ordering comparisons, the *Timestamp* value is treated as a 64-bit unsigned integer, with *Seconds* as the most significant.

Typed Name

Description

This syntax is used by attributes whose values represent a level and an interval associated with an object.

Syntax ID

#define SYN_TYPED_NAME 25

Data Formats

Transfer Syntax

```
uint32      Length
uint32      Level
uint32      Interval
unicode     Distinguished Name
```

API Data Structure

```
typedef  struct
{
    char        *objectName;
    uint32      level;
    uint32      interval;
}   Typed_Name_T;
```

Matching Rules

Equality

Used In

Notify
Partition Control
Print Server
Printer
Queue

Remarks

This syntax names an NDS object and attaches two numeric values to it:

- The *Level* of the attribute indicates the priority
- The *Interval* represents the number of seconds between certain events or the frequency of reference

The *Distinguished Name* identifies the NDS object referred to by the *Typed Name*.

More specific interpretation of the two values depends on the context in which the attribute value is used. The values are user-assigned and are relative. To be effective the user must implement them. The user can use these values to implement iterative intervals or to enforce priorities. All three parts of the field must be equal for attributes to be equal.

Unknown

Description

This syntax represents strings of binary information. This syntax is used by attributes whose attribute definition has been deleted from the schema.

Syntax ID

#define SYN_UNKNOWN 0

Data Formats

Transfer Syntax

```
uint32      Length = N
BYTE[N]     Content
```

API Data Structure

```
typedef  struct
{
    char        *attrName;
    uint32      syntaxID;
    uint32      valueLen;
    void        *value;
}   Unknown_Attr_T;
```

Matching Rules

none

Used In

Unknown

Remarks

The syntax representation on the wire is the same as an *Octet String*. The *Content* field has no additional intrinsic semantics.