Novell Developer Kit

www.novell.com

NOVELL EDIRECTORY™ CORE SERVICES

June 2008

Novell®

Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export, or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. Please refer to www.novell.com/info/exports/ for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.novell.com/company/legal/patents/ and one or more additional patents or pending patent applications in the U.S. and in other countries.

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

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

Novell Trademarks

For Novell trademarks, see the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide

This book describes how to access Novell® eDirectory™ services and store information in its tree, and how to access and modify the information types that it can store. This information is divided into the following sections:

Feedback

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

Documentation Updates

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

Additional Information

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

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

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

For product information about eDirectory, see the eDirectory Documentation Site (http://www.novell.com/documentation/edir88/).

Documentation Conventions

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

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

Programming Concepts

1

This chapter describes programming concepts, how groups of functions interact with each other to provide the functionality you need to use Novell® eDirectory™ in your applications. You should already be familiar with eDirectory, its architecture, and its services. For this information, see *NDK: Novell eDirectory Technical Overview*.

1.1 Context Handles

Most eDirectory functions have a context handle as the first parameter. A context handle is similar to a directory handle in that it points to a specific location, but it is different in that a directory handle points to a location in the file system and a context handle points to a location in the eDirectory tree. As with directory handles, applications can have multiple context handles.

A context handle knows its location in the eDirectory tree by maintaining the distinguished name of the location. In addition to the context name, the context handle maintains the following types of information:

- Tree name
- Current connection
- Name format
- Settings for dereferencing aliases, working in Unicode, using types, and canonicalizing names.

This section describes the following aspects of context handles:

- Section 1.1.4, “DCK_CONFIDENCE Key,” on page 20
- Section 1.1.5, “DCK_NAME_CONTEXT Key,” on page 20
- Section 1.1.6, “DCK_LAST_CONNECTION Key,” on page 21
- Section 1.1.7, “DCK_TREE_NAME Key,” on page 21
- Section 1.1.8, “DCK_DSI_FLAGS Key,” on page 21
- Section 1.1.9, “DCK_NAME_FORM Key,” on page 22
- Section 1.1.10, “DCK_NAME_CACHE_DEPTH Key,” on page 23
- Section 1.1.11, “Multi-Threaded Applications,” on page 23

For step-by-step instructions, see

- “Creating a Context Handle” on page 49
- “Freeing a Context Handle” on page 50
- “Modifying the Context of the Context Handle” on page 50

For sample code, see ndscontx.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm).

1.1.1 Management of Context Handles

The following functions are used to manage context handles.

Function	Description
NWDSCreateContextHandle (page 133)	Creates a context handle and initializes it with default values.
NWDSDuplicateContextHandle (page 144)	Creates a copy of an existing context handle and initializes it with the values of the existing handle.
NWDSFreeContext (page 160)	Frees a previously allocated context handle.

When a context handle is created, it is initialized to the following default values:

- DCK_FLAGS (DCV_DEREF_ALIASES | DCV_XLATE_STRINGS | DCV_CANONICALIZE_NAMES). With these values, the eDirectory libraries dereference aliases, translate Unicode strings to the local code page, and append the name context to the object names. The libraries expect applications to submit partial names, relative to the name context, and in the character set of the local code page. The libraries return partial names, with the name context stripped off, and in the character set of the local code page.
- DCV_NAME_CONTEXT. For workstation applications, the name context is the default eDirectory context used during login. For NLMs, this is the server's bindery context. If multiple contexts are listed, it is the first context in the list.
- DCK_CONFIDENCE (DCV_LOW_CONF). The default value allows information to be obtained from all replica types (read-only, read-write, and master).
- DCK_LAST_CONNECTION. This key is initialized to no connection (-1).
- DCK TREE NAME. For workstation applications, this is the preferred tree of the workstation. For NLMs, this is the tree the server belongs to.

- DCK_DSI_FLAGS (DSI_ENTRY_FLAGS | DSI_OUTPUT_FIELDS | DSI_SUBORDINATE_COUNT | DSI_MODIFICATION_TIME | DSI_BASE_CLASS | DSI_ENTRY_RDN | DSI_ENTRY_DN). This key determines the type of entry information that can be obtained about an object. The default value returns information about the type of object (for example, alias, partition, or container object), the number of objects subordinate to the specified object, the object's modification timestamp, the base class of the object, the name of the object (default format is partial name with partial dot form), and the distinguished name of the object (default format is partial dot form).
- DCK_NAME_FORM (DCV_NF_PARTIAL_DOT). This key specifies the name form; the default is partial dot.
- Name cache depth. This key specifies how many names will be stored in memory. The default is five.

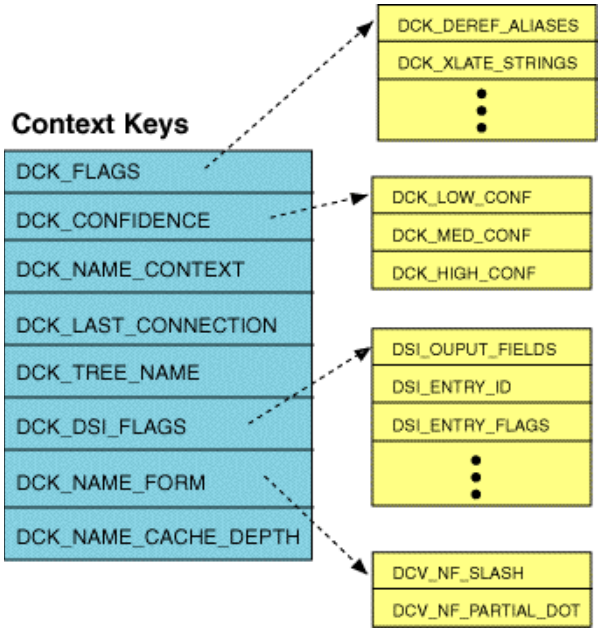
These settings determine the format of the name that the eDirectory libraries expect eDirectory functions to use and the format of the name the functions return to the application.

For example, suppose a workstation logs in to the eDirectory tree with a default name context of “Clerks.Accouting.ACME” and a username of “JRoss.” With these default settings, the libraries would append “Clerks.Accouting.ACME” to any object name passed as a parameter in an eDirectory function.

If an “HPPrinter” object exists in a “Printer” container under “Clerks,” the function should use “HPPrinter.Printer” for the printer's name. A dot should be the delimiter because that is the default name form.

The following graphic illustrates how these keys interact with their flags, bit masks, or keys.

Figure 1-1 *Context Keys Interacting with Flags, bit Masks, and Keys*



Since all of these keys can be modified, the next section describes the functions that you use to manage the context handle settings.

1.1.2 Modification of Context Handle Settings

The following functions are used to modify the context handle keys and to obtain information about the context handle and its settings.

NWDSGetContext (page 191)	Reads the information about the context.
NWDSSetContext (page 387)	Modifies the information maintained by the context handle.

To read the context handle settings or to modify them, you need to understand the context keys. They are described in greater detail in the following sections.

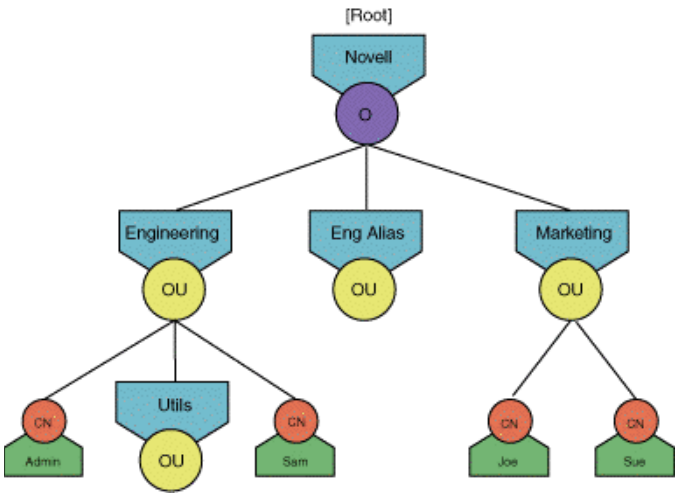
1.1.3 DCK_FLAGS Key

The DCK_FLAGS key holds a bit mask of flags that can be ORed together. Each flag affects how the eDirectory libraries return object name information.

DCV_DEREF_ALIASES. This flag determines whether an object name, which references an alias, returns information about the alias object or the object that it references. The default is to return information about the referenced object. However, if your application needs to read the values of the alias's attributes, then this flag needs to be turned off (set to 0).

In search and list operations, this flag determines whether the object, that is submitted as the start point for the search, can be dereferenced. For example, the eDirectory tree in the figure below has an alias object, Eng Alias, that references the Engineering container.

Figure 1-2 *DCV_DEREF_ALIASES Flag*



If Eng Alias is submitted as the object for the start of the search or list, it will be dereferenced and return information about the Engineering container and its subordinate objects.

For search functions, another parameter, usually called searchAliases, determines whether subordinate objects are dereferenced.

DCV_XLATE_STRINGS. This flag determines the types of strings the eDirectory libraries return and expect to receive. Natively, eDirectory works in Unicode. However, if this flag is on (set to 1),

the libraries expect strings in the local code page and translate between Unicode and the local code page.

If this flag is turned off (set to 0), the libraries do not translate and expect Unicode strings. This can improve performance.

DCV_TYPELESS_NAMES. This flag determines whether the libraries return names in typeless (JRoss.HR.ACME) or typeful (CN=JRoss.OU=HR.O=ACME) format. When this flag is off (which is the default value), the libraries return typeful names. You can either set this flag to the way you want to display eDirectory names to your user, or you can use name functions to modify the name before displaying it.

The following functions manipulate the form of the name.

NWDSAbbreviateName (page 80)	Converts a name to its shortest, typeless form relative to a specified name context.
NWDSCanonicalizeName (page 114)	Converts an abbreviated name to the canonical form (fully distinguished, with types).
NWDSRemoveAllTypes (page 353)	Removes all attribute types from a distinguished name.

Be aware that the NWDSCanonicalizeName function adds types to the name if the DCV_TYPELESS_NAME flag is off. To add types, it must use the default typing rule that makes some assumptions about the containers in the eDirectory tree. When applying types, the libraries make the following assignments:

- The most significant (right-most) component is an Organization (O).
- The least significant (left-most) component is a Common Name (CN).
- All intervening components are Organizational Units (OU).

For example, if you pass in a name such as “JRoss” with a name context of “Engineering.ACME” to the NWDSCanonicalizeName function, the function returns “CN=JRoss.OU=Engineering.O=ACME”. The library does not know the correct types to apply to the container names and cannot look them up; it simply follows the default typing rules.

If the eDirectory tree contains Tree Root, Country, or Locality containers, the types will be inaccurate. To have the server resolve a typeless name and return a correctly typed distinguished name, use the NWDSReadObjectDSIInfo (page 338) function and set the DSI flags to return only the distinguished name.

DCV_CANONICALIZE_NAMES. A name in canonical form is a name that is fully distinguished; it can have types, but it isn't required to. eDirectory does all of its name processing with fully distinguished names.

When this flag is clear, the libraries expect input of fully distinguished names and return fully distinguished names. When this flag is set, the libraries expect input of partial names and append the name context to the partial name to create a distinguished name. On return values, the libraries strip off the name context and return partial names. For example:

Name passed in:	JRoss
Current Context:	HR.ACME
Resulting Name:	JRoss.HR.ACME

The libraries use the following rules when determining how to expand a name:

- A period preceding a name prevents the libraries from appending the context to the name. The libraries assume that such a name is a distinguished name.
- For each trailing period, the libraries remove one component from the name context before appending it to the name.

If you place a period at the beginning of the name passed in, the libraries treat it as 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.Engineering..
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 passed in was only 22 characters. Both examples resulted in the same name, but they used different rules to achieve that name.

DCV_DEREF_BASE_CLASS. This flag allows eDirectory to return the base class of the object the alias references. When this flag is turned off, eDirectory returns the base class of the alias, which is always the Alias class.

DCV_DISALLOW_REFERRALS. This flag allows eDirectory to refer a request to another server and the request will follow such referrals until the information is found. If this flag is turned on, the request must be answered by the first eDirectory agent it is sent to. If this agent does not have a replica with the information, the request fails.

1.1.4 DCK_CONFIDENCE Key

The DCK_CONFIDENCE key determines where the libraries can obtain eDirectory information. Most of the time, eDirectory information can be obtained from any type of replica. However, a few operations, especially partition operations, must be performed on the master replica. You can force eDirectory to obtain information from the master replica by setting the DCK_CONFIDENCE key to DCV_HIGH_CONF. Since eDirectory cannot use the first replica that it finds, requesting information to come only from master replicas can slow down performance.

1.1.5 DCK_NAME_CONTEXT Key

This key determines the NDS context that is added to partial names to make them distinguished names when the DCV_CANONICALIZE_NAME flag is set. If your application changes context, you need to set this key to the new context.

1.1.6 DCK_LAST_CONNECTION Key

This key contains the connection that was last used to service an eDirectory request. Since an eDirectory request can require connections to a server other than the server that first receives the request, this value changes during the processing of the request.

1.1.7 DCK_TREE_NAME Key

This key contains the name of the eDirectory tree. If your application allows logins to multiple trees, you should create a context handle for each tree and set this key to that tree's name.

1.1.8 DCK_DSI_FLAGS Key

The DCK_DSI_FLAGS key is a bit mask that determines what entry information is returned about objects. Some information types, such as DSI_CREATION_TIMESTAMP or DSI_PARENT ID, apply to all object types, but some are particular to an object type, such as DSI_REPLICA_TYPE, which applies only to Partition objects. Each of these DSI flags are described in the following sections.

DSI_OUTPUT_FIELDS. This flag determines what is returned. It is a bit mask of all the other DSI flags. Only those flags ORed into the bit mask have information returned. The information is returned in the same order that flags were ORed together.

Not all versions of eDirectory can supply the requested information. If the request goes to an eDirectory server that cannot supply the information, the server clears the corresponding bit in the DSI_OUTPUT_FIELDS flag.

Before attempting to read DSI information, you should always read the DSI_OUTPUT_FIELDS flag to determine what was actually returned.

DSI_ENTRY_ID. This flag returns the entry ID of the object. All objects in the server's local eDirectory database have an entry ID that is specific to that database.

DSI_ENTRY_FLAGS. This flag returns information about the state of entry. These identify significant characteristics about the entry, such as whether the entry is an alias object, a partition root, a container, a container alias, or an audited entry. All that apply to the entry are ORed together. For a complete list, see Section 5.12, “DSI_ENTRY_FLAGS Values,” on page 473.

DSI_MODIFICATION_TIME. This flag returns the time of the last modification to the entry. The time is returned as the number of seconds since 12:00 midnight, 1 January 1970. If the modification time is unknown, returns 0.

DSI_MODIFICATION_TIMESTAMP. This flag returns the timestamp of the last modification of the entry. A timestamp includes the number of seconds since 12:00 midnight, 1 January 1970 as well as the replica number where the modification took place and the eDirectory event type that identifies the type of modification. If the timestamp is unknown, it returns 0.

DSI_REVISION_COUNT. This flag returns the number of times the entry has been modified.

DSI_CREATION_TIMESTAMP. This flag returns the timestamp that indicates when the entry was created. The timestamp includes the number of seconds since 12:00 midnight, 1 January 1970 as well as the replica number where the creation occurred and an eDirectory event type that indicates that this is a creation event. If the creation timestamp is unknown, it returns 0.

DSI_BASE_CLASS. This flag returns the base class, or object class, that was used to create the entry. Although Object Class is an attribute of every class and the eDirectory libraries include functions to read attribute information, this flag can be used to obtain base class information without the overhead of a special request to read the Object Class attribute.

DSI_ENTRY_RDN. This flag returns the partial name of the entry, in the format specified by the DCK_FLAGS and the DCK_NAME_FORM keys. The default values for these keys would return the name of the entry with the name context stripped, in dot form, and without types.

DSI_ENTRY_DN. This flag returns the distinguished name of the entry in the format specified by the DCK_FLAGS and DCK_NAME_FORM keys. The default value for these keys would return the name in dot form and without types.

DSI_PARENT_ID. This flag returns the entry ID of the entry's parent object.

DSI_PARENT_DN. This flag returns the distinguished name of the entry's parent object in the format specified by the DCK_FLAGS and DCK_NAME_FORM keys. The default value for these keys would return the name in dot form and without types.

DSI_DEREFERENCE_BASE_CLASS. This flag returns the base class of the object an alias references if the DCV_DEREF_BASE_CLASS flag is set. If this flag is not set, it returns the base class of the Alias entry, which is always the Alias object class.

DSI_SUBORDINATE_COUNT This flag returns the number of objects that are subordinate to the entry. If this number is unknown, it returns 0.

DSI_PARTITION_ROOT_ID. This flag returns the entry ID of the partitions' root container.

DSI_PARTITION_ROOT_DN. This flag returns the distinguished name of the partition's root container in the format specified by the DCK_FLAGS and DCK_NAME_FORM keys. The default value for these keys would return the name in dot form and without types.

DSI_PURGE_TIME. This flag returns the oldest purge time for a Partition object. The time is the number of seconds since 12:00 midnight, 1 January 1970.

DSI_REPLICA_TYPE. This flag returns the replica's type. See Section 5.23, “Replica Types,” on page 483 for a list.

DSI_REPLICA_NUMBER. This flag returns the number the replica was assigned when it was created. This number is unique among the replicas of the partition and is used to identify the replica where an eDirectory event occurred.

DSI_REPLICA_STATE. This flag returns the current state of the replica. See Section 5.24, “Replica States,” on page 484 for a list.

1.1.9 DCK_NAME_FORM Key

The eDirectory libraries support two name forms: partial dots and slashes. The partial dot form uses dots as delimiters, with the root-most object at the right, and does not include the tree name. For example, the following name uses the default, partial dot form:

```
JRoss.HR.ACME
```

The slash form uses back slashes as delimiters, with the root-most object at the left, and includes the tree name. For example, the same name in the XYZ_tree could be displayed as

```
\XYZ_tree\ACME\HR\JRoss
```

The DCK_NAME_FORM key determines the input and output form for the eDirectory libraries:

- If slash form is turned on, names are always returned as fully distinguished and must be entered as fully distinguished. Since slash form names are always distinguished, the libraries ignore the setting of the DCV_CANONICALIZE_NAMES flag.
- If partial dot form is turned on, names can be either partial or fully distinguished, depending on the setting of the DCV_CANONICALIZE_NAMES flag. If this flag is turned on, partial names can be entered.

The following figure summarizes the interaction of the DCK_NAME_FORM key with the DCV_CANNOICALIZE_NAMES flag and the DCV_TYPELESS_NAMES flag.

Figure 1-3 *The DCK_NAME_FORM Key Interacts with the DCV_CANNOICALIZE_NAMES Flag and the DCV_TYPLESS_NAMES Flag*

Name Form	Canonical Flag On?		Typeless Flag On?	
Slash	Yes	N/A	Yes	Typeless Distinguished Name
	No	N/A	No	Typeful Distinguished Name
Partial Dot	Yes	Partial Name	Yes	Typeless Partial Name
	No	Distinguished Name	No	Typeful Distinguished Name

N/A: Not Applicable

This key also affects a number of the DSI flags that specify the return of the object's RDN, the object's DN, the DN of the object's parent, and the DN of the partition root.

1.1.10 DCK_NAME_CACHE_DEPTH Key

Name caching is a feature that allows a context handle to store a specified number of names in memory. This enhancement was made to increase performance of the resolve name operations. When the cache is full, the oldest name is dropped to add a new name.

To clear the name cache, set the depth of the cache to zero by calling NWDSSetContext (page 387) (context, DCK_NAME_CACHE_DEPTH, &0). To reinitialize name cache, set the cache depth to the desired value by calling NWDSSetContext (page 387) (context, DCK_NAME_CACHE_DEPTH, &5) again.

Currently, the default depth of the name cache is five. To set the depth to a lower value than the current setting, you must clear the cache and then set it to the desired depth.

1.1.11 Multi-Threaded Applications

Context handles are designed to enable the NWDS functions to operate correctly in a multi-threaded application. The library stores persistent state information for eDirectory operations in the context handle. As a result, the functions are thread-safe as long as threads do not share context handles. If

an application shares context handles among threads, the application is responsible for providing proper concurrency locks for shared resource.

1.2 Buffer Management

Applications access eDirectory 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 eDirectory functions pass input and output parameters through specially defined local buffers. Therefore, an understanding of buffer management is essential to developing eDirectory-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 sections describe buffer size, initialization, message length, allocation types, required search buffers, and then list the functions used in buffer management.

- Section 1.2.1, “Buffer Size in eDirectory,” on page 24
- Section 1.2.2, “Initialization Operations for eDirectory Input Buffers,” on page 25
- Section 1.2.3, “eDirectory Buffer Allocation and Initialization Functions,” on page 25
- Section 1.2.4, “eDirectory Input Buffer Functions,” on page 25
- Section 1.2.5, “eDirectory Output Buffer Functions,” on page 26

See Also:

- “Preparing eDirectory Input Buffers” on page 52
- “Preparing eDirectory Output Buffers” on page 53
- “Retrieving Results from eDirectory Output Buffers” on page 53
- “Freeing eDirectory Buffers” on page 53

1.2.1 Buffer Size in eDirectory

To allocate an input or output buffer, call NWDSAllocBuf (page 95). 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.

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. The DEFAULT_MESSAGE_LEN constant is defined as 4 KB.

1.2.2 Initialization Operations for eDirectory Input Buffers

Only input buffers (also called request buffers) need to be initialized with an operation type. Output buffers (also called result buffers) do not need to be initialized. Input buffers are used in read and search operations to restrict the operation to a list of classes or attributes. They are also used when creating class or attribute definitions and when adding objects to or modifying objects in the eDirectory tree. See Section 5.3, "Buffer Operation Types and Related Functions," on page 464 for a list of operation types for buffer initialization.

Once the buffer is initialized, it is ready to receive data. However, you cannot put data directly into the buffer. eDirectory has a set of specialized functions for placing the data into the input buffer.

1.2.3 eDirectory Buffer Allocation and Initialization Functions

The following functions are used to allocate input and output buffers and to initialize the input buffers.

NWDSAllocBuf (page 95)	Allocates a Buf_T structure and the requested number of bytes.
NWDSInitBuf (page 242)	Initializes a Buf_T structure for input.
NWDSFreeBuf (page 158)	Destroys a Buf_T structure and frees the memory allocated to it.

1.2.4 eDirectory Input Buffer Functions

The following functions are used to set up the input or request buffers with the information that limits the scope of the request from all items of a type to the items listed in the request buffer.

NWDSBeginClassItem (page 110)	Begins the insertion of a class definition into an input buffer.
NWDSPutAttrName (page 308)	Inserts the name of an attribute into an input buffer.
NWDSPutAttrVal (page 312)	Inserts an attribute value into an input buffer.
NWDSPutAttrNameAndVal (page 310)	Inserts the name of an attribute and its value into an input buffer.
NWDSPutChange (page 314)	Inserts a change record into an input buffer.
NWDSPutChangeAndVal (page 316)	Inserts a change record and an attribute value into an input buffer.
NWDSPutClassItem (page 319)	Inserts a class item into an input buffer.
NWDSPutClassName (page 321)	Inserts the name of an object class into an input buffer.
NWDSPutSyntaxName (page 325)	Inserts the name of a syntax into an input buffer.

NWDSPutFilter (page 323)	Inserts a search filter expression tree into an input buffer.

1.2.5 eDirectory Output Buffer Functions

The following functions are used to retrieve information from output or result buffers.

NWDSComputeAttrValSize (page 130)	Returns the size of the next attribute value in a result buffer.
NWDSGetAttrCount (page 169)	Returns the number of attributes in a result buffer.
NWDSGetAttrDef (page 171)	Returns the next attribute definition in a buffer.
NWDSGetAttrName (page 173)	Returns the next attribute and the number of associated values in a result buffer.
NWDSGetAttrVal (page 175)	Returns the next value of an attribute in a result buffer.
NWDSGetClassDef (page 183)	Returns the next class definition in a result buffer.
NWDSGetClassDefCount (page 185)	Returns the number of class definitions in a result buffer.
NWDSGetClassItem (page 187)	Returns the next item of a class item list in a result buffer.
NWDSGetClassItemCount (page 189)	Returns the number of items in a class item list in a result buffer.
NWDSGetObjectCount (page 210)	Returns the number of objects in a result buffer.
NWDSGetObjectName (page 214)	Returns the next object name in a result buffer.
NWDSGetPartitionInfo (page 224)	Returns the next partition information structure in a result buffer.
NWDSGetServerName (page 234)	Returns the name of the server and the number of partition names in a result buffer.
NWDSGetSyntaxCount (page 236)	Returns the number of syntax definitions in a result buffer.
NWDSGetSyntaxDef (page 238)	Returns the next syntax definition in a result buffer.

1.3 Read Requests for Object Information

The purpose of eDirectory is to store information in the form of objects that can be accessed easily across the network. eDirectory provides functions for

- Listing objects
- Reading an object's attributes, attribute values, and extended information
- Comparing attribute values

This section describes how to determine what objects are available and how to retrieve the information stored in them. For information on reading schema definitions, see Section 1.13, “Read Requests for Schema Information,” on page 45.

For sample code, see the following:

- ndsbrows.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- ndsreada.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- readinfo.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

1.3.1 eDirectory List Operations

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 (page 248)	Lists the objects that are subordinate to a specified object.
NWDSListByClassAndName (page 254)	Lists objects of a particular object class and/or object name that are subordinate to a specified object.
NWDSListContainers (page 261)	Lists container objects that are subordinate to a specified object.
NWDSExtSyncList (page 146)	Lists the immediate subordinates of an object placing restrictions on the names, classes, modification times, and object types of the subordinates.

Each list function provides a method for restricting the list so that it does not include every object in the eDirectory database. Even restricted, many of these functions return more information than the output buffer can hold. These functions use an iteration handle. See the next section for information on using the handle and retrieving all the information.

1.3.2 Controlling Iterations

The iterationHandle parameter controls the retrieval of output data that is larger than the output buffer pointed to by the subordinates parameter in the NWDSList function. Several eDirectory functions (for example, NWDSRead, NWDSReadReferences, and NWDSSearch) use an iteration handle, and they all called it an iterationHandle parameter (the output buffer name varies with the function). Before the initial call to an iterative function, 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 (page 126) to free memory associated with the operation.

1.3.3 Retrieving Object Information from the Output Buffer

Since there is no way to know what a read or list function returns to an output buffer, most of these functions have specialized functions for retrieving the data from the buffer. For example, the NWDSList function returns a list of objects. You would use the following functions to retrieve the information from its output buffer:

NWDSGetObjectCount	Returns the number of objects in the buffer.
NWDSGetObjectName	Returns a name of an object in the buffer. Must be called for each object.

If the function returns other information, such as attributes or attributes and values, other functions are provided to find out how many attributes are associated with the object or how many values are associated with an attribute.

1.3.4 eDirectory Read Operations

When you read eDirectory objects, you 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, or Telephone Number attributes. However, the eDirectory database maintains other information about an object, such as when it was created and last modified. You can use the following functions to read information about the object, its attributes, and extended information.

NWDSRead (page 327)	Reads the object's attribute information. This is configurable and may include attribute names only, attributes and values, effective privileges, attribute value flags, and attribute creation and modification timestamps. For more information, see Section 5.16, "Information Types for Search and Read," on page 476.
NWDSReadObjectInfo (page 340)	Reads object information not stored in the attributes of the object. This includes the base class used to create the object, class flags, the modification time, and the number of objects that are subordinate to this object.
NWDSExtSyncRead (page 150)	Reads values from one or more attributes of the specified object, allowing restrictions on modification time.

NWDSReadObjectDSIInfo (page 338)	Returns DSI object information not stored in the attributes of an object.This is configurable and may include information such as object creation and modification timestamps, subordinate object count, and the entry ID of the parent object. If the object is a replica, it may return the entry ID of the partition's root entry, the replica type, purge time, and replica number. For more information, see Section 5.6, “Context Keys and Flags,” on page 467.
NWDSReadReferences (page 342)	Searches all the replicas on a particular server and returns any objects that contain attributes that reference the specified object. This function can be configurable to also return attribute values. For more information, see Section 5.16, “Information Types for Search and Read,” on page 476
NWDSReadNDSInfo (page 336)	Reads eDirectory server information from the server the connection handle specifies. This is function can be configurable to return the following information: the number of levels the server is located from the root container, eDirectory version number, OS version, OS name, current UTC time of the server, and the server's eDirectory tree. For more information, see Section 5.19, “eDirectory Ping Flags,” on page 479.

1.3.5 Configuring Results

NWDSRead (page 327) is a representative of eDirectory functions that are configurable. (Others include NWDSSearch (page 383), NWDSReadClassDef (page 333), and NWDSReadAttrDef (page 330).) These functions let you configure the results to be returned.

The eDirectory functions use two methods to configure return results: function parameters and context handle flags.

Function Parameters. Function parameters usually include an infoType parameter that specifies the type of information to return, a boolean parameter that specifies all or only listed items, and a parameter that contains the list of the requested items. For example, the NWDSRead function uses an infoType parameter to specify what attribute information to return, an allAttrs parameter to specify whether all attributes are returned or only specified attributes, and an attrNames parameter which is used to list specific attributes.

If the allAttrs parameter is TRUE, eDirectory returns information for all attributes of the object and the attrNames parameter is ignored. Therefore, the attrNames parameter should be set to NULL. If the allAttrs parameter is FALSE, eDirectory returns the attributes named in the input buffer, which is pointed to by the attrNames parameter.

If the allAttrs parameter is FALSE and the attrNames parameter is NULL, no attribute information is returned. However, you can use the return value of the NWDSRead (page 327) function to determine whether the object exists or whether access to the object is allowed.

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.

Context Handle Flags. The flags configure the context handle. Since most eDirectory functions have a context handle parameter, most eDirectory functions are influenced by configuration changes to the context handle. For more information on configuring the context handle, see Section 1.1, "Context Handles," on page 15.

1.3.6 Attribute Value Comparisons

NWDSCompare (page 128) compares a given value with the values assigned to a specified attribute. For example, you could ask eDirectory to compare whether the Member attribute of a particular group is equal to the name of some User object. 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.

1.4 Search Requests

eDirectory provides a powerful and useful searching capability that allows you to retrieve information based on your specified searching criteria. For example, you can perform a search that allows your application to

- Find all users in the HR department who have a salary above 35,000
- Find all the printers that are located on the third floor

A search request must include not only search criteria, but also the area to be searched and the amount of information to be returned for each matching object. The sections below describe the following aspects of setting up search requests:

Remember, if you are creating a client application, you must initialize the Unicode table with either the NWCallsInit or the NWInitUnicodeTables function before starting the search.

Also, you must have a valid context handle that you have set to the container in the eDirectory tree from which you want the search to begin. The context handle's keys and flags determine how the search handles aliases and how object names are returned. For more information, see Section 1.1, "Context Handles," on page 15.

For a code example, see ndssearc.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm).

1.4.1 Buffers Needed for eDirectory Searches

The NWDSSearch (page 383) function requires three buffers: two input buffers (called NameBuffer and FilterBuffer in this example) and an output buffer (called ResultBuffer). The purpose of these buffers is summarized in the list that follows.

NameBuffer	Restricts the search results to certain attributes only. A Name Buffer is not required if you want to return all attribute information or no attribute information.
FilterBuffer	Contains the search expression. If no filtering is needed, construct a filter equivalent to objectclass=*.
ResultBuffer	Stores the search results.

The FilterBuffer determines what eDirectory searches for. The NameBuffer determines what information is returned about the objects that match the search criteria. The ResultBuffer stores this information.

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 is Buf_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. The following code makes a request to return a match when the Surname and Telephone attributes have values.

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

FilterBuffer is initialized to the DSV_SEARCH_FILTER operation:

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

Finally, you should allocate a filter cursor which initializes 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);
```

1.4.2 Search Filter Components

You search for information in eDirectory by first building a search filter and then putting the filter into the filter buffer. The search filter describes the set of conditions that satisfy the search. To create a search filter, use the following functions:

NWDSAllocFilter (page 97)	Allocates space for the filter.

NWDSAddFilterToken (page 84)	Adds search criteria to the allocated filter.
NWDSPutFilter (page 323)	Places the finished search filter into an input filter buffer. This function normally frees the space that has been allocated for the filter.
NWDSFreeFilter (page 162)	Frees the space allocated for the filter if the NWDSPutFilter (page 323) function is not called or fails.

When you allocate a search filter by calling the NWDSAllocFilter (page 97) function, you receive a filter cursor. The filter cursor is initialized to the current insertion point. It is empty until you add search filter information. The cursor is used as input as you add the search filter information.

A search filter is made up of a tree of filter nodes. Each node consists of the following three parts:

- Token
- Value
- Syntax

Token. The token determines what can be in the other parts of the node. A token can be one of the following:

FTOK_ANAME	Signals that an attribute name is in the node.
FTOK_AVAL	Signals that an attribute value is in the node.
Relational Operator	Usually comes before an attribute name or value node and signals how the search treats the following node. For example, signals whether the search is looking for a value equal to or greater than the following node.
Logical Operator	Usually comes before an attribute name or value node and signals how the search treats the following node. For example, signals whether the next node is another condition that must be met or whether the condition is an either one or the other relationship.
FTOK_END	Signals the end of the search filter.

Value. Value only applies when the token is FTOK_ANAME or FTOK_AVAL. When one of these is the token, the value specifies the name of the attribute or the value, respectively. For all other tokens, value must be set to NULL.

Syntax. Syntax only applies where the token is FTOK_ANAME or FTOK_AVAL. Otherwise, it must be set to 0 (zero). When the node contains an attribute value or name, the syntax contains the syntax ID of the attribute. Use the NWDSGetSyntaxID (page 240) function to obtain the syntax ID.

Logical Operators. The logical operator tokens express logical relationships among attribute value assertions. The following table shows the logical operators and the conditions that test TRUE for each.

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.

You can control precedence among the logical operators by inserting tokens that act as parentheses. In the absence of parentheses, the FTOK_AND operator takes precedence over the FTOK_OR operator, and the FTOK_NOT operator takes precedence over both.

Relational Operators. 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. A relational operator must be followed by a node that asserts the test case, for example an FTOK_ANAME or FTOK_AVAL.

The following table shows the relational operators and the conditions that test TRUE for each.

Token	Value	Comment
FTOK_EQ	7	TRUE only if the attribute's value is equal to the asserted value. Must be followed by a FTOK_AVAL node that contains the value. For example, to set up a search where the attribute must equal an integer value of 5, use the following node expressions: FTOK_EQ, NULL, 0 FTOK_AVAL, "5", SYN_INTEGER
FTOK_GE	8	TRUE only if the attribute's value is greater than or equal to the asserted value. Must be followed by a FTOK_AVAL node that contains the value. See FTOK_EQ for a sample syntax.
FTOK_LE	9	TRUE only if the relative ordering places the asserted value before any of the attribute's values. Must be followed by a FTOK_AVAL node that contains the value. See FTOK_EQ for a sample syntax.
FTOK_APPROX	10	TRUE only if the value of the attribute matches the asserted value. If the attribute syntax does support approximate match, this operator matches for equality. Must be followed by a FTOK_AVAL node that contains the value. See FTOK_EQ for a sample syntax.

Token	Value	Comment
FTOK_PRESENT	15	TRUE only if the named attribute is present in the entry. Must be followed by a FTOK_ANAME node that contains the attribute's name. For example, to set up a search where the object must have a Given Name attribute, use the following node expressions: FTOK_PRESENT, NULL, 0 FTOK_ANAME, "Given Name", SYN_CI_STRING
FTOK_RDN	16	TRUE only if the object's Relative Distinguished Name matches the asserted value. Must be followed by a FTOK_ANAME node that contains the RDN. May be used in a search without authentication. For example, to set up a search that returns all objects that start their name with D, use the following node expressions: FTOK_RDN, NULL, 0 FTOK_ANAME, "D*", SYN_DIST_NAME
FTOK_BASECLS	17	TRUE only if the object belongs to the asserted base class. Must be followed by a FTOK_ANAME node that contains the name of the base class. May be used in a search without authentication. For example, to set up a search where the object must be a Group object, use the following node expressions: FTOK_BASECLS, NULL, 0 FTOK_ANAME, "Group", SYN_CLASS_NAME
FTOK_MODTIME	18	TRUE only if the modification time stamp is greater than or equal to the asserted value.
FTOK_VALTIME	19	TRUE only if the creation time stamp 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 back slash escape character to escape the asterisk (*) or to escape the back slash itself (\\).

1.4.3 Sample 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 (page 84).

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:

```
Base Class=User AND Surname=Sm*
```

To create the expression tree, think of each element as corresponding to a node in the tree. The sequence for adding the nodes is the same as if you were processing the string from left to right. You add one node for each element and add one node to signal the end:

Element	Node Expression
Object's Base Class	(FTOK_BASECLS, NULL, 0)
User	(FTOK_AVAL, "User", SYN_CLASS_NAME)
AND	(FTOK_AND, NULL, 0)
Surname	(FTOK_ANAME, "Surname", SYN_CI_STRING)
=	(FTOK_EQ, NULL, 0)
Sm*	(FTOK_AVAL, "Sm*", SYN_CI_STRING)
	(FTOK_END, NULL, 0)

Once you form a search expression, you build the expression tree by calling the NWDSAddFilterToken (page 84) function to add each node to the tree. Each token represents a node on the expression tree:

The following example demonstrates how to build a search filter that looks for user objects with a Surname not equal to “brown.” It starts with a string expression for the filter.

```
Expression:
   NOT (Surname EQ "brown")

Code Example:
   NWDSAllocFilter(&FilterCursor);
   NWDSAddFilterToken(FilterCursor, FTOK_NOT, NULL,0);
   NWDSAddFilterToken(FilterCursor, FTOK_LPAREN, NULL,0);
   NWDSGetSyntaxID(context,"surname",&syntaxID);
   NWDSAddFilterToken(FilterCursor, FTOK_ANAME,"surname", syntaxID);
   NWDSAddFilterToken(FilterCursor, FTOK_EQ, NULL,0);
   NWDSAddFilterToken(FilterCursor, FTOK_AVAL,"brown", syntaxID);
   NWDSAddFilterToken(FilterCursor, FTOK_RPAREN, NULL,0);
   NWDSAddFilterToken(FilterCursor, FTOK_END,NULL,0);
```

After you have added all of the filter tokens to the cursor, your next task is to store the filter expression in the filter buffer.

```
NWDSPutFilter (context, filterBuffer, FilterCursor, FreeValuePointer);
```

The FreeValuePointer parameter can be passed either as NULL or as a pointer to a function that frees the attribute values.

If the NWDSPutFilter function succeeds, the FilterCursor is automatically freed. If the NWDSPutFilter function fails, you should call the NWDSFreeFilter function to free the FilterCursor. Do not call the NWDSFreeFilter function when the NWDSPutFilter function succeeds or you will crash your program.

Once you have put the search filter in the filter buffer, you are ready to start the search by calling the NWDSSearch (page 383) function.

1.4.4 Retrieving Information from the Result Buffer

When the search completes, you cannot just read the information. You must retrieve the information from the result buffer. Remember to pull out all the information you requested, whether you need it or not. Use the infoType, allAttrs, and attrNames parameters in the NWDSSearch function to control what is returned. The attrNames parameter has been called the NameBuffer in the previous sections.

If you requested attribute names and values, you cannot retrieve just attribute names. The failure to pull out the attribute values makes it impossible to retrieve the rest of the attribute names. Use the following functions:

NWDSGetObjectCount	Returns the number of objects whose information is stored in the result buffer.
NWDSGetObjectName	Returns the name of the current object and the count of attributes associated with the object.
NWDSGetAttrName	Returns attribute's name and the number of values associated with the attribute.
NWDSComputeAttrValSize	Returns the size of the attribute's value. This function is only required if you don't know the size of the attribute.
NWDSGetAttrVal	Returns the attribute's value. Must be called for each value associated with the attribute.

For each object in the result buffer, you must retrieve all the attributes, and all the values for each attribute, before you can retrieve information about the next object in the result buffer.

A search request can return more information than can fit in the result buffer. To retrieve all the information, you need to call the NWDSSearch function repeatedly until the iterationHandle parameter is equal to NO_MORE_ITERATIONS. For more information on iteration handles, see “Controlling Iterations” on page 27.

1.4.5 Search Cleanup

After you have retrieved all the information you need from the result buffer, you should clean up all the buffers you allocated. Use the following functions:

NWDSFreeBuf (FilterBuffer)
NWDSFreeBuf (ResultBuffer)
NWDSFreeBuf (NameBuffer)
NWDSLogout (context)
NWDSFreeContext (context)
NWFreeUnicodeTable ()

1.5 Developing in a Loosely Consistent Environment

eDirectory 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 updated instantaneously, 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 one second or as long as five minutes.

For a developer who is new to the eDirectory environment, this can present many new challenges. A program that works well in the test environment can suddenly become unreliable when installed on a real network. 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. The following topics detail some things you need to consider:

- Section 1.5.1, "Loose Consistency," on page 37
- Section 1.5.2, "Disappearing eDirectory Objects," on page 37
- Section 1.5.3, "Disappearing eDirectory Objects: Solutions," on page 38

1.5.1 Loose Consistency

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

Loose consistency has the advantage of allowing eDirectory servers to be connected to the network with different types of media. For example, a company might connect parts of its network 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 eDirectory 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.

1.5.2 Disappearing eDirectory Objects

eDirectory guarantees a loosely consistent Directory, meaning that changes made to one replica may take time to synchronize with all other replicas. It is possible that a user or other object may access one of these other replicas before these replicas receive the updated information. For example, a program might create an object in the eDirectory tree and then collect the data it needs to modify the object's attributes. The operations used to gather the data might change the context variable that points to the specific replica where the object was created.

Now that the program has the data it needs, the program will call NWDSModifyObject (page 286), which then walks the tree to find the partition holding the object. If the replica it contacts has not yet received the information it needs to create the object, the operation will return a NO_SUCH_ENTRY (-601) error.

1.5.3 Disappearing eDirectory Objects: Solutions

There are several ways to avoid or correct the problem of disappearing objects. The simplest solution is to wait until after the next synchronization interval. If you don't want to wait, you could call NWDSSyncPartition (page 396), which signals the synchronization engine to update the specified partition without waiting for 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 (page 226) 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);
```

Another solution would be 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.

You might also create a separate context handle for operations that deal with other objects. This would ensure that the context handle for the object that was just added would not be changed.

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, which 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. Note that this 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.

1.6 Add Object Requests

The NWDSAddObject (page 87) function is used to add nodes to an NDS™ directory. If those 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 and the name you are going to give it, and to have a buffer prepared containing all required data for the object. The NWDSAddObject function has the following syntax:

```
NWDSCCODE N_API NWDSAddObject(
   NWDSContextHandle   context,          /* (IN) Indicates the
                                            eDirectory context
                                            for the request.*/
   pnstr8             *objectName,       /* (IN) Points to the name
                                            of the object to be
                                            added.*/
   pnint_ptr          *iterationHandle,  /* (IN) */
   nbool8              more,             /* (IN) */
   pBuf_T              objectInfo);      /* (IN) Points to a request
                                            buffer containing the
                                            attributes 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.

The NWDSAllocBuf (page 95) function is used to allocate an eDirectory 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 the size parameter of this function: DEFAULT_MESSAGE_LEN (4069) and MAX_MESSAGE_LEN (64512). The NWDSAllocBuf (page 95) function has the following syntax:

```
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.*/
```

The NWDSInitBuf (page 242) function is used to initialize an eDirectory buffer for an eDirectory 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 NWDSInitBuf function has the following syntax:

```
NWDSCCODE N_API NWDSInitBuf(
    NWDSContextHandle   context,    /*(IN) Indicates the eDirectory
                                           context for the request.*/
    nuint32             operation,  /*(IN) Indicates the eDirectory
                                           operation for which
                                           the buffer is being
                                           initialized.*/
    pBuf_T              buf);       /*(IN) Points to the buffer being
                                           initialized.*/
```

When calling NWDSInitBuf (page 242), you must specify the context handle and the buffer. You must also specify the intended use (allocation type) of the buffer in the operation parameter. See Section 5.3, “Buffer Operation Types and Related Functions,” on page 464 for a list of operation types. For the NWDSAddObject function, the operation type is DSV_ADD_ENTRY.

After the buffer has been created and initialized, you are ready to add the information to the buffer to create the new object. This is done by calling the NWDSPutAttrName (page 308) and NWDSPutAttrVal (page 312) 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 */
```

The NWDSPutAttrName (page 308) function has 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 in which to store
                                        the attribute name.*/
    pnstr8              attrName); /*(IN) Points to the attribute
                                        name to store in the
                                        request buffer.*/
```

The NWDSPutAttrVal (page 312) function has the following syntax:

```
NWDSCCODE N_API NWDSPutAttrVal(
    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 the attribute value.*/
    nptr                attrVal); /* (IN) Points to the attribute
                                        value to be stored in the
                                        request buffer.*/
```

The syntaxID parameter contains the data type of the attribute value. This data type is defined for each attribute in "Base Attribute Definitions". For instance, for the Object Class attribute, the sample code inputs the value User and the type SYN_DIST_NAME. The attribute values and syntax IDs are found in the "Attribute Syntax Definitions". If you are trying to add a User, for example, "Base Object Class Definitions" defines the object "User". These definitions (in *NDK: Novell eDirectory Schema Reference*) list all of the mandatory attributes as well as the optional attributes.

Next, for each of the attributes you want to set, go to "Base Attribute Definitions" which contains an entry for each attribute type; for example, the "CN (Common Name)" attribute. The definitions specify the syntax (for CN, this is Case Ignore String).

"Attribute Syntax Definitions" defines the "Case Ignore String"syntax. 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.

The buffer must include the attribute name and its value for all mandatory attributes. Names and values for optional attributes can be added to the buffer or added after the object is created.

NOTE: Although the naming attribute for the object is always a mandatory attribute, do not add this attribute to the buffer. The objectName parameter of the NWDSAddObject function takes the value for one naming attribute. If the object has multiple naming attributes, use the objectName parameter for one attribute and add the others to the buffer.

Once you have the buffer set up with the appropriate attributes and values, you are ready to call NWDSAddObject (page 87).

See Also:

- "Adding an eDirectory Object" on page 58
- ndsadd1.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

1.7 eDirectory Security and Applications

This module describes how you, as a software developer, can use eDirectory 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 are called [Entry Rights]
- Properties are called Attributes
- All Properties Rights are called [All Attributes Rights]

Most of Security programming is very simple when you know how to read and write information about objects in the eDirectory tree. Reading and writing objects and their information is discussed in detail in other chapters, so the focus here is on what is specific to security.

Let's look at what a program would do to read the Access Control Lists (ACLs) of a particular object. Once you become familiar with eDirectory 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()            This function initializes the Unicode tables
                         and low-level interface functions.

NWDSCreateContextHandle()   Get a context allocated for our program to
                             communicate with eDirectory

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?

   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
      Object_ACL_T, which holds all of the ACL information we need.
      We can process this information to 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 allocated earlier

NWDSFreeUnicodeTables() Now we've done all the eDirectory
housecleaning items
```

You use the standard functions NWDSModifyObject (page 286) and NWDSRead (page 327) 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 (page 451) structure. This is the structure that holds the contents of the ACL. The structure type definition is:

```
typedef struct
{
 pnstr8       protectedAttrName;
 pnstr8       subjectName;
 nuint32      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.
- When you use [All Attributes Rights] as the attribute name, you are giving (or reading) the rights to the entire set of attributes.

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 [All Attributes Rights], which indicates Joe has rights to all properties, or attributes. The subjectName would be "Joe.Sales.MyCompany", to indicate the user who 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 (see Section 5.18, "eDirectory Access Control Rights," on page 477).

1.8 Authentication of Client Applications

Most client workstations log in when they are started. This login process establishes an authenticated connection to eDirectory and a licensed connection to the network server to which you are attached. A licensed connection gives your application rights, such as rights to a particular file system volume, to perform eDirectory operations.

If your application is meant to run on a client workstation, you can either assume that a licensed connection has been established, or you can log in to establish a licensed connection. In either case, your development 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 (page 297) and then authenticate the connection using NWDSAuthenticateConn (page 103).

NWDSAuthenticateConn (page 103) authenticates and licenses the connection so that you can access the file system.

1.9 Multiple Tree Support

eDirectory Libraries for C provide multiple tree support. Your application can have the user log in to each tree, and then the libraries can manage the background authentication as the user accesses resources in each eDirectory tree.

Client applications and NLM applications use different methods.

- NLM applications manipulate multiple tree identities through the current user in the thread group structure. See Section 1.9.1, "NLM Applications and Multiple Tree Identities," on page 43 for more information.
- Client applications manipulate multiple tree identities through the context handle. See Section 1.9.2, "Client Applications and Multiple Tree Identities," on page 44 for more information.

The libraries supply the following functions for discovering eDirectory trees.

Function	Description
NWDSScanConnsForTrees (page 379)	Scans existing connections and returns tree names.
NWDSScanForAvailableTrees (page 381)	Scans the bindery of the specified connection and returns one tree name. To return multiple tree names, the function must be called multiple names.
NWDSReturnBlockOfAvailableTrees (page 376)	Scans the bindery of the specified connection and returns the specified number of tree objects.

1.9.1 NLM Applications and Multiple Tree Identities

An NLM has a current user associated with each thread group. The eDirectory Libraries for C use the current user for background authentication. If an NLM needs to establish connections to multiple eDirectory trees or use multiple identities to log in to the same eDirectory tree, the NLM must manage the current user in the thread group structure. Two functions, NWDSGetCurrentUser and NWDSSetCurrentUser, allow this manipulation.

An NLM creates a new identity by first calling the NWDSSetCurrentUser function with identity parameter set to zero (0). This removes the current user from the thread group structure. The NLM then calls NWDSCreateContextHandle. The eDirectory libraries check the thread group structure for a current user. Since there isn't one, the eDirectory libraries create a new current user for the thread group. The last step is a call to NWDSLogin which authenticates and establishes credentials for this new user.

The NWDSGetCurrentUser function allows the NLM to get the current user information so that information can be saved and used again with that identity. The NWDSSetCurrentUser function allows that identity to be restored to a thread group.

If an NLM wants to switch between two or more users, the NLM must save and manage the user information for each user created. The eDirectory libraries do not save the user information for multiple users. They maintain only information about the current user in the thread group structure. If the NLM does not save and manage the information for multiple users, the information is lost. For step-by-step instructions for accessing multiple eDirectory trees, see "Establishing Identities to Multiple eDirectory Trees—NLM Platform" on page 55.

1.9.2 Client Applications and Multiple Tree Identities

Applications for Windows 95, Windows 98, and Windows NT establish identities to multiple eDirectory trees by manipulating the DCK_TREE_NAME key in the context handle.

For step-by-step instructions for accessing multiple eDirectory trees, see "Establishing Identities to Multiple eDirectory Trees—Client Platforms" on page 56.

1.10 Effective Rights Function

These two functions are specific to security.

Function	Description
NWDSGetEffectiveRights (page 203)	Returns the effective rights of one object to access another object or its attributes.
NWDSListAttrsEffectiveRights (page 251)	Returns the effective rights of one object to access another object's attributes.

See Also:

- readeff.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

1.11 Partition Functions

These functions operate on partitions.

Function	Comment
NWDSJoinPartitions (page 246)	Joins a subordinate partition to its parent partition.
NWDSListPartitions (page 264)	Lists the immediate subordinates of an object.
NWDSListPartitionsExtInfo (page 267)	Returns information about the replicas of partitions stored on a specified server.
NWDSRemovePartition (page 361)	Deletes an existing partition by deleting its master replica.
NWDSSplitPartition (page 394)	Divides a partition into two partitions at a specified object.

Function	Comment
NWDSSyncPartition (page 396)	Signals the synchronization process to schedule an update of a specified partition a specified number of seconds into the future.

1.12 Replica Functions

These functions operate on replicas of a partition.

Function	Comment
NWDSAddReplica (page 91)	Adds a replica of an existing partition to a server.
NWDSChangeReplicaType (page 122)	Changes the replica type of a given replica on a given server.
NWDSRemoveReplica (page 363)	Deletes a replica from the replica set of a partition.
NWDSSyncReplicaToServer (page 398)	Requests a replica to initiate synchronization with a specified server.

1.13 Read Requests for Schema Information

The schema read functions work similar to the read functions for database entry (object) information. They use input and output buffers, information types, iteration handles if the result buffer may not be able to contain all the requested information, and specialized functions for retrieving the information from the result (output) buffer. See Section 1.3, “Read Requests for Object Information,” on page 26 if you are not familiar with these procedures.

Class Definition Functions. The following table lists the functions that you can use to read information about object class definitions.The list includes the functions that read the information and the specialized functions that retrieve the information from the result buffer.

Function	Purpose
NWDSReadClassDef (page 333)	Reads a class definition. This function is configurable so that you can control the type of information it returns about a class. All requested information about a class must be retrieved before moving to the next class.
NWDSListContainableClasses (page 258)	Returns the names of the object classes that the specified object class can contain.
NWDSGetClassDefCount (page 185)	Returns the number of class definitions in the result buffer.
NWDSGetClassDef (page 183)	Retrieves the name of an object class and, if requested, the class flags and ASN.1 identifier.

Function	Purpose
NWDSGetClassItemCount (page 189)	Returns the number of items of a particular information type. Must be called for each information type specified in the read request. For example, the DS_CLASS_DEFS information type returns five types of information: super classes, containment classes, naming attributes, mandatory attributes, and optional attributes. This function must be called once for each of these types, but only after all the information items of the previous type has been retrieved.
NWDSGetClassItem (page 187)	Returns a class information item. Must be called repeatedly for each item in the count. For example, if the class has two super classes, it must be called twice to retrieve the two super class names.

For step-by-step instructions for reading schema information, see the following:

- "Reading a Class Definition" on page 70
- "Listing Containable Classes" on page 69

For sample code, see rdclsdef.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm).

Attribute Definition Functions. The following table lists the functions that you can use to read attribute type definitions. The list includes the functions that read the information and the specialized functions that retrieve the information from the result buffer.

Function	Purpose
NWDSReadAttrDef (page 330)	Reads existing attribute type definitions.
NWDSGetAttrCount (page 169)	Returns the number of attributes whose information is stored in a result buffer.
NWDSGetAttrDef (page 171)	Returns the next attribute definition from a result buffer.

For step-by-step instructions on reading attribute definitions, see "Reading an Attribute Definition" on page 72.

For sample code, see rdattdef.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm).

Attribute Syntax Definition Functions. The following table lists the functions that you can use to read information about syntaxes and their definitions. The list includes the read functions and the specialized functions that retrieve the information from the result buffer.

Function	Purpose
NWDSReadSyntaxDef (page 346)	Returns the syntax definition for the specified syntax ID.
NWDSGetSyntaxID (page 240)	Returns the syntax ID for the specified attribute.

Function	Purpose
NWDSReadSyntaxes (page 348)	Returns a list of attribute syntax definitions. You can configure this function to return just names or names and definitions.
NWDSGetSyntaxCount (page 236)	Returns the number of syntaxes whose information is stored in a result buffer filled by the NWDSReadSyntaxes function.
NWDSGetSyntaxDef (page 238)	Returns the next syntax definition from a result buffer filled by the NWDSReadSyntaxes function.

For step-by-step instructions on reading syntax definitions, see "Retrieving Syntax Names and Definitions" on page 73.

1.14 Schema Extension Requests

The schema can be extended by adding new attributes and classes. New syntax definitions cannot be added. New attributes must be added before the attributes can be added to the class.

1.14.1 Attribute Definition Functions

The following table lists the functions that you can use to add or delete the attribute type definitions in the schema.

Function	Purpose
NWDSDefineAttr (page 135)	Creates a new attribute type definition.
NWDSRemoveAttrDef (page 355)	Deletes attribute type definitions. Attributes can be deleted only if they are not assigned to an object class and if they are not part of the operational schema.

Adding attributes is usually only the first step in the process of extending the schema. An attribute cannot be used by a class until it is added to the class when the class is first defined or if the class is already defined, when the class is modified.

Attributes cannot be deleted if they are used by a class definition.

For step-by-step instructions, see

- "Creating an Attribute Definition" on page 68
- "Deleting an Attribute Definition" on page 69

1.14.2 Class Definition Functions

The following table lists the functions that you can use to add or modify the object class definitions in the schema.

Function	Purpose
NWDSDefineClass (page 137)	Creates a new class.
NWDSModifyClassDef (page 281)	Modifies an existing class.
NWDSRemoveClassDef (page 357)	Deletes a class definition.

These functions are used in conjunction with other specialized functions. The input buffer, which will contain the information for the new class or a modification for an existing class, must be allocated and then initialized for the operation. (For the operation type, see Section 5.3, "Buffer Operation Types and Related Functions," on page 464.) The information must be placed in the buffer using specialized functions. Each information item is placed in the buffer separately. For example, to add three optional attributes to a new class definition, you would call NWDSPutClassItem once for each attribute. After all the information has been placed in the input buffer, you call the function that modifies or creates the class.

Your input buffer must be big enough to hold all the information to create a class or modify it.

For step-by-step instructions, see

- "Creating a Class Definition" on page 67
- "Modifying a Class Definition" on page 70
- "Deleting a Class Definition" on page 69

For code samples, see

- crclsdef.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- rdclsdef.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

Tasks

2

This chapter describes the most common tasks associated with accessing the Novell® eDirectory™ tree, its information, and schema.

2.1 Context Handle Tasks

An NDS context points to a specific location in the eDirectory tree. This section gives instructions on how to create, free, modify, and read a context.

2.1.1 Creating a Context Handle

To create a context handle, the Unicode tables must be initialized.

1 Call NWDSCreateContextHandle (page 133).

The following code illustrates these steps.

```
NWDSCCODE            ccode;
NWDSContextHandle    context;
ccode = NWDSCreateContextHandle(&context);
if(ccode)
    /* handle creation error */
```

See Also:

- “Reading the Context of the Context Handle” on page 51
- “Modifying the Context of the Context Handle” on page 50
- “Freeing a Context Handle” on page 50

- ndscontx.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

2.1.2 Freeing a Context Handle

Freeing a context frees memory.

1 Call NWDSFreeContext (page 160).

The following code illustrates this procedure.

```
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);
```

See Also:

- ndscontx.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- "Creating a Context Handle" on page 49

2.1.3 Modifying the Context of the Context Handle

The following steps explain how to modify the context of the context handle. (For a list of other keys and flags that can be modified, see Section 5.6, "Context Keys and Flags," on page 467.)

Modifying the context of the context handle is only important if the context handle is set to use both dot name forms and canonicalized names. (These values are assigned as defaults when a context handle is created.)

1 Call NWDSSetContext (page 387) with the DCK_NAME_CONTEXT flag and the name of the new context.

The context name should be in the following format:

- In the local code page if the SCV_XPLATE_STRINGS flag is on (default) or Unicode if off.
- The complete name of the context, without the tree name.

The following code illustrates these procedures.

```
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.");
}
```

See Also:

- "Freeing a Context Handle" on page 50
- "Reading the Context of the Context Handle" on page 51
- Section 5.6, "Context Keys and Flags," on page 467
- ndscontx.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

2.1.4 Reading the Context of the Context Handle

The following steps explain how to read the context of the context handle. (For a list of other information that can be read about the context handle, see Section 5.6, "Context Keys and Flags," on page 467.)

1. Declare a variable that matches the data type of the context name key (a NULL terminated string) and that is large enough to hold the name of the context.

   Local code page names need a length of MAX_DN_CHARS+1 and Unicode names need ((MAX_DN_CHARS+1)*2).

2. Call NWDSGetContext (page 191) with the key parameter set to DCK_NAME_CONTEXT.

   The following code illustrates these procedures.

```
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\nNWDSGetContext returned error <%d>",err);
   }
   else
   {
      printf("\nCurrent Name Context: %s",name);
   }
}
```

See Also:

- "Modifying the Context of the Context Handle" on page 50
- "Freeing a Context Handle" on page 50
- Section 5.6, "Context Keys and Flags," on page 467
- ndscontx.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

2.2 Buffer Tasks

Many eDirectory functions require input buffers or output buffers. Many require both. This section provides procedures for managing these buffers.

- Section 2.2.1, “Preparing eDirectory Input Buffers,” on page 52
- Section 2.2.2, “Preparing eDirectory Output Buffers,” on page 53
- Section 2.2.3, “Retrieving Results from eDirectory Output Buffers,” on page 53
- Section 2.2.4, “Freeing eDirectory Buffers,” on page 53

2.2.1 Preparing eDirectory Input Buffers

This task prepares a memory buffer for writing data to eDirectory.

1. Allocate memory for the input buffer by calling NWDSAllocBuf (page 95). 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 (page 242). Choose one of the Initialization Operations for eDirectory Input Buffers (page 25) 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 eDirectory Input Buffer Functions (page 25) 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.* eDirectory 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.

See Also:

- “Preparing eDirectory Output Buffers” on page 53
- “Freeing eDirectory Buffers” on page 53

2.2.2 Preparing eDirectory Output Buffers

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

1 Allocate memory for the input buffer by calling NWDSAllocBuf (page 95). 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. eDirectory contains a complete set of eDirectory Output Buffer Functions (page 26) 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.

See Also:

- "Retrieving Results from eDirectory Output Buffers" on page 53
- "Preparing eDirectory Input Buffers" on page 52

2.2.3 Retrieving Results from eDirectory Output Buffers

This task reads data from an output buffer.

1 Determine the number of objects in the buffer by calling NWDSGetObjectCount (page 210).

2 Determine the amount of memory required for each object attribute by calling NWDSComputeAttrValSize (page 130).

3 Allocate memory to receive the attribute data.

4 Retrieve the attribute data from the buffer by calling the eDirectory Output Buffer Functions (page 26) 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.

See Also:

- "Freeing eDirectory Buffers" on page 53

2.2.4 Freeing eDirectory Buffers

This task releases a buffer allocated by NWDSAllocBuf (page 95). After you have executed an operation using a buffer and retrieved the information from the output buffer (if applicable), always free the buffer memory.

1 To free buffer memory, call NWDSFreeBuf (page 158), as shown in the following example.

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

See Also:

- "Listing Objects in an eDirectory Container" on page 60

2.3 Authentication and Connection Tasks

To access all but the public available eDirectory information, applications must establish a connection and authenticate to eDirectory. This section provides instructions on accessing eDirectory information that is available from the connection, authenticating to eDirectory, retrieving the server address of the connection.

- Section 2.3.1, "Accessing eDirectory Ping Information," on page 54
- Section 2.3.2, "Authenticating to eDirectory," on page 54
- Section 2.3.3, "Establishing Identities to Multiple eDirectory Trees—NLM Platform," on page 55
- Section 2.3.4, "Establishing Identities to Multiple eDirectory Trees—Client Platforms," on page 56
- Section 2.3.5, "Retrieving Addresses of a Connected Server," on page 57

2.3.1 Accessing eDirectory Ping Information

1. Allocate a request buffer of type Buf_T (page 436) by calling NWDSAllocBuf (page 95).
2. Call NWDSReadNDSInfo (page 336). Pass the requestedFields parameter an OR of DSPING flags for which information is needed (see Section 5.19, "eDirectory Ping Flags," on page 479).
3. Call NWDSGetNDSInfo (page 208) to retrieve information returned about any single field. Pass in the result buffer returned from NWDSReadNDSInfo for the resultBuffer parameter, a single flag for the requestedField parameter, and a pointer to memory of the appropriate size for the data parameter.

NOTE: For NWDSGetNDSInfo, fields can be called in any order, and the information can be retrieved so long as the buffer has not been reused or freed. A particular field can even be retrieved multiple times.

2.3.2 Authenticating to eDirectory

Most client workstations log in to the network when they are booted making it unnecessary for many client applications to perform this task. See Section 1.8, "Authentication of Client Applications," on page 42.

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

1. Initialize an eDirectory context by calling NWDSCreateContextHandle (page 133).
2. If needed, call NWDSSetContext (page 387) to change context values.

For information about changing your context, see Section 1.1, “Context Handles,” on page 15 and “Modifying the Context of the Context Handle” on page 50.

3 Log in to eDirectory by calling NWDSLogin (page 270).

4 Open a new connection by calling either NWDSOpenConnToNDSServer (page 297), NWCCOpenConnByName (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk645.html), or NWCCOpenConnByRef (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk661.html).

5 Authenticate and license the new connection by calling NWDSAuthenticateConn (page 103).

NOTE: The process of authenticating to eDirectory 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 eDirectory as “admin”; it is authenticated as “public”. If you want to do anything with eDirectory other than read public information, you must log in. The authentication credentials are stored on the thread group level and are accessible only by the OS.

2.3.3 Establishing Identities to Multiple eDirectory Trees—NLM Platform

NLMs establish identities to multiple eDirectory trees by manipulating the DCK_TREE_NAME key and by managing the current user in the thread group structure. NLMs must manage the current user associated with each thread group.

To establish identities to two eDirectory trees, follow these steps. They are separated into three procedures: logging in User 1 to Tree 1, logging in User 2 to Tree 2, and switching between users.

User 1 to Tree 1

To log in User 1 to Tree 1, follow these steps.

1 Call NWDSSetCurrentUser (page 389) with the userHandle parameter set to zero (0) to clear the user information in the thread group structure.

2 Call NWDSCreateContextHandle (page 133).

3 Call NWDSSetContext (page 387) with DCK_TREE_NAME as the value for the key parameter and point the value parameter to the name of Tree 1.

These instructions call this the context handle for User 1.

4 Call NWDSLogin (page 270) with the context handle for User 1, the eDirectory name for User 1, and User 1's password.

5 Call NWDSGetCurrentUser (page 196) and save the information to use for User 1 in Tree 1.

User 2 to Tree 2

To log in User 2 to Tree 2, follow these steps.

1 Call NWDSSetCurrentUser (page 389) with the userHandle parameter set to zero (0) to clear the user information in the thread group structure.

2 Call NWDSCreateContextHandle (page 133).

3 Call NWDSSetContext (page 387) with DCK_TREE_NAME as the value for the key parameter and point the value parameter to the name of Tree 2.

   These instructions call this the context handle for User 2.

4 Call NWDSLogin (page 270) with the context handle for User 2, the eDirectory name for User 2, and User 2's password.

5 Call NWDSGetCurrentUser (page 196) and save the information to use for User 2 in Tree 2.

Switching Between Trees

With two users authenticated to two eDirectory trees, you can switch between them by following these steps.

1 Call NWDSSetCurrentUser (page 389) with the userHandle parameter set to the value for User 1.

2 Use the context handle that you set up for User 1 in Tree 1 and call the eDirectory functions to perform work in Tree 1.

3 To switch to User 2 in Tree 2, call NWDSSetCurrentUser (page 389) with the userHandle parameter set to the value for User 2.

4 Use the context handle that you set up for User 2 in Tree 2 and call the eDirectory functions to perform work in Tree 2.

2.3.4 Establishing Identities to Multiple eDirectory Trees—Client Platforms

Applications for Windows 95, Windows 98, and Windows NT establish identities to multiple eDirectory trees by manipulating the DCK_TREE_NAME key. To establish an identity to two eDirectory tree, follow these steps. They are divided into three tasks: logging User 1 in to Tree 1, logging User 2 in to Tree 2, and switching between users.

User 1 to Tree 1

To log User 1 in to Tree 1, follow these steps.

1 Call NWDSCreateContextHandle (page 133) to create a context handle.

2 Call NWDSSetContext (page 387) with DCK_TREE_NAME as the value for the key parameter and point the value parameter to Tree 1.

   These instructions call this the context handle for User 1.

3 Call NWDSLogin (page 270) with the context handle for User 1, the eDirectory name for User 1, and User 1's password.

User 2 to Tree 2

To log in User 2 to Tree 2, follow these steps.

1 Call NWDSCreateContextHandle (page 133) to create a context handle.

2 Call NWDSSetContext (page 387) with DCK_TREE_NAME as the value for the key parameter and point the value parameter to Tree 2.

   These instructions call this the context handle for User 2.

3 Call NWDSLogin (page 270) with the context handle for User 2, the eDirectory name for User 2, and User 2's password.

Switching between Trees

With two users authenticated to two eDirectory trees, you can switch between them by following these steps.

1 To perform work on Tree 1, call eDirectory functions with the context handle for User 1.

2 To perform work on Tree 2, call eDirectory functions with the context handle for User 2.

2.3.5 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 (page 95). This buffer does not need to be initialized since it is a result buffer.

2 Call NWDSGetServerAddresses2 (page 230).

3 Call NWDSComputeAttrValSize (page 130) to find the size of the address data in the buffer.

4 Allocate a contiguous block of memory the size of the attribute value, and set a void pointer to point to that block.

5 Call NWDSGetAttrVal (page 175), passing in the pointer to the allocated memory.

6 When NWDSGetAttrVal (page 175) returns, typecast the pointer to be a pointer to Net_Address_T, and retrieve the information.

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

8 Loop to Step 3 until all addresses have been removed from the result buffer.

9 Free the result buffer by calling NWDSFreeBuf (page 158).

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

See Also:

- “Preparing eDirectory Input Buffers” on page 52

2.4 Object Tasks

After authenticating to an eDirectory tree, the most common tasks are those associated with adding information to the tree (objects and attribute values), modifying the information, and reading (searching and comparing) the existing information. This section provides procedures for these kinds of tasks:

- Section 2.4.6, “Listing Objects in an eDirectory Container,” on page 60
- Section 2.4.7, “Modifying an eDirectory Object,” on page 61
- Section 2.4.8, “Adding an Auxiliary Class to an eDirectory Object,” on page 62
- Section 2.4.9, “Reading Attributes of eDirectory Objects,” on page 62
- Section 2.4.10, “Searching eDirectory,” on page 63

2.4.1 Adding an eDirectory Object

To add an object to eDirectory, follow these steps:

1. Allocate memory for the request buffer by calling NWDSAllocBuf (page 95).
2. Set the iterationHandle to NO_MORE_ITERATIONS.
3. Initialize the request buffer for a DSV_ADD_ENTRY (7) operation by calling NWDSInitBuf (page 242).
4. For each attribute to be supplied for the object, first store the attribute’s name in the result buffer by calling NWDSPutAttrName (page 308). Then store the associated value(s) in the result buffer by calling NWDSPutAttrVal (page 312) 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 (page 87).
6. Free the request buffer by calling NWDSFreeBuf (page 158).

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

See Also:

- ndsadd1.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- Section 1.6, “Add Object Requests,” on page 38

2.4.2 Comparing Attribute Values

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

1. Allocate a request buffer by calling NWDSAllocBuf (page 95).
2. Initialize the request buffer for a DSV_COMPARE operation by calling NWDSInitBuf (page 242).
3. Place the name of the attribute whose value you want to compare into the request buffer by calling NWDSPutAttrName (page 308).
4. Place the value you want to compare into the buffer by calling NWDSPutAttrVal (page 312).
5. Compare the values by calling NWDSCompare (page 128).
6. Check matched to see if the values matched.
7. Free the request buffer by calling NWDSFreeBuf (page 158).

See Also:

- "Searching eDirectory" on page 63

2.4.3 Deleting an eDirectory Object

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

1. To delete an Object, call NWDSRemoveObject (page 359)

   This function 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.

See Also:

- "Determining the Effective Rights of an Object" on page 59
- readeff.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- deluser.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

2.4.4 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 (page 95). 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 (page 95).
4. Initialize the request buffer for a DSV_READ operation by calling NWDSInitBuf (page 242).
5. Place the attribute names in the request buffer by calling NWDSPutAttrName (page 308) once for each attribute name.
6. Set iterationHandle to NO_MORE_ITERATIONS.
7. Call NWDSListAttrsEffectiveRights (page 251).
8. Determine the number of attributes in the result buffer by calling NWDSGetAttrCount (page 169).
9. For each attribute in the result buffer, retrieve the information by calling NWDSGetAttrVal (page 175).
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 (page 158).
12. Free the result buffer by calling NWDSFreeBuf (page 158).

eDirectory 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 (page 251), 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" on page 27.

See Also:

- "Finding the Host Server of an Object" on page 60

2.4.5 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 (page 95). This buffer does not need to be initialized since it is a result buffer.
2. Call NWDSGetObjectHostServerAddress (page 212).
3. Call NWDSGetAttrCount (page 169) to determine the number of attributes stored in the result buffer.
4. Call NWDSGetAttrName (page 173) to retrieve the attribute name (network address) and the count of attribute values.
5. For each attribute value, call NWDSComputeAttrValSize (page 130) 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 (page 175) and passing in the pointer allocated in Step 5.
8. When NWDSGetAttrVal (page 175) 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.

See Also:

- "Reading Attributes of eDirectory Objects" on page 62

2.4.6 Listing Objects in an eDirectory Container

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

1. Allocate memory for the output buffer by calling NWDSAllocBuf (page 95).
2. Set the iteration handle to NO_MORE_ITERATIONS.
3. Call NWDSList (page 248).
4. Determine the number of subordinate objects in the output buffer by calling NWDSGetObjectCount (page 210).

5 Call NWDSGetObjectName (page 214) 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 (page 158).

Aborting Iterative Operations

If you need information on aborting an iterative operation, see “Controlling Iterations” on page 27.

See Also:

- ndsbrows.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- “eDirectory List Operations” on page 27

2.4.7 Modifying an eDirectory Object

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

1 Allocate a data buffer by calling NWDSAllocBuf (page 95).

2 Initialize the buffer for a DSV_MODIFY_ENTRY (9) operation by calling NWDSInitBuf (page 242).

3 For each attribute value to be modified, place the desired changes into the buffer using NWDSPutChange (page 314) and NWDSPutAttrVal (page 312).

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

   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 (page 286) once.

   On multivalued attributes

   - If you want to remove a value and add a new value, you must place two changes in the request buffer: one to remove the old value using the DS_REMOVE_VALUE flag and one to add the new value with the DS_ADD_VALUE flag.
   - If you want to ensure the attribute has a value without triggering synchronization, use NWDSCompare (comparison operations are faster than read or write operations).
   - If you want to ensure the attribute has a value, its timesstamp is updated, and synchronization is triggered, use NWDSModifyObject with the DS_OVERWRITE_VALUE flag as the changeType.

   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 (page 95) and initialized by calling NWDSInitBuf (page 242). 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 Section 5.5, “Change Types for Modifying Objects,” on page 466.

4 Make the modification by calling NWDSModifyObject (page 286).

Remember, if an attempt is made to modify the Object Class attribute on NDS 7.xx and below, an error is returned. NDS 8 and above allows modifications to the Object Class attribute (see “Adding an Auxiliary Class to an eDirectory Object” on page 62).

See Also:

- ndsmodob.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

2.4.8 Adding an Auxiliary Class to an eDirectory Object

Auxiliary class are only supported on servers running NDS 8 or higher. If you connect to a server running an earlier version of eDirectory that has a replica with the object, you cannot added the auxiliary class to the object because the Object Class attribute cannot be modified. To add an auxiliary class, you must use a connection to a server that is running NDS 8 and that also has a replica with the object.

The following steps assume that you have connected to an NDS 8 server and that the auxiliary class is defined in the eDirectory schema. Auxiliary classes are added to the schema just like other schema definitions. (For more information, see “Creating a Class Definition” on page 67.)

To add an auxiliary class to a User object, complete the following steps.

1. Allocate a data buffer by calling NWDSAllocBuf (page 95).
2. Initialize the buffer for a DSV_MODIFY_ENTRY (9) operation by calling NWDSInitBuf (page 242).
3. Use the NWDSPutChange (page 314) function to put the Object Class attribute in the data buffer and NWDSPutAttrVal (page 312) function to put the name of the auxiliary class in the data buffer.

   Auxiliary classes can have mandatory attributes. If the class you added in Step 3 has mandatory attributes, you must put the attribute name and its value into the same data buffer that adds the auxiliary class to the Object Class attribute. Use the functions listed in Step 3 to add any mandatory attributes.

   Values for optional attributes can be added now or later.
4. Add the auxiliary class by calling NWDSModifyObject (page 286).

You can add and delete only auxiliary class values from the Object Class attribute. The values that define the object's base class and its super classes are protected so that they cannot be deleted.

2.4.9 Reading Attributes of eDirectory Objects

Read the attributes of an object by following these steps:

1. Allocate memory for the output buffer by calling NWDSAllocBuf (page 95).
2. If you are requesting information for all attributes, skip to Step 6.
3. Call NWDSAllocBuf (page 95) to allocate memory for the input buffer.
4. Call NWDSInitBuf (page 242) to initialize the input buffer for a DSV_READ operation.
5. Call NWDSPutAttrName (page 308) once for each attribute name being placed in the input buffer.
6. Set iteration handle to NO_MORE_ITERATIONS.

7 Set infoType to DS_ATTRIBUTE_VALUES.

8 Call NWDSRead (page 327).

9 Call NWDSGetAttrCount (page 169) to determine the number of attributes in the output buffer.

10 Call NWDSGetAttrName (page 173) to retrieve the name of the current attribute and the number of its values from the output buffer.

11 Call NWDSComputeAttrValSize (page 130) to determine the size of the attribute value; then allocate a block of memory that size.

12 Call NWDSGetAttrVal (page 175) to read the attribute value. For multivalued attributes, call this function for each of the values.

13 Free the memory that was allocated in Step 11.

14 If all of the attribute information has not been read from the output buffer, loop to Step 10.

15 If the iteration handle is not equal to NO_MORE_ITERATIONS, loop to Step 8.

16 If an input buffer was allocated, free the input buffer by calling NWDSFreeBuf (page 158).

17 Free the output buffer by calling NWDSFreeBuf (page 158).

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 (page 173) will return erroneous information.

Aborting Iterative Operations

If you need information on aborting an iterative operation, see “Controlling Iterations” on page 27.

See Also:

- ndsreada.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- Section 1.3, “Read Requests for Object Information,” on page 26
- “eDirectory Read Operations” on page 28

2.4.10 Searching eDirectory

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

Follow these steps to search a region of eDirectory:

1 Call NWDSAllocBuf (page 95) 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 (page 95) to allocate the request buffer.

4 Call NWDSInitBuf (page 242) to initialize the request buffer for a DSV_SEARCH operation.

5 Call NWDSPutAttrName (page 308) once for each attribute name to place the names of the desired attributes into the request buffer.

6 Call NWDSAllocFilter (page 97) to allocate a filter expression tree.

7 Call NWDSAddFilterToken (page 84) once for each search token to place the search-filter conditions in the expression tree.

   To select all objects, use the filter

   ```
   FTOK_BASECLS *
   ```

8 Call NWDSAllocBuf (page 95) to allocate a filter buffer.

9 Call NWDSInitBuf (page 242) to initialize the request buffer for a DSV_SEARCH_FILTER operation.

10 Call NWDSPutFilter (page 323) to store the search-filter expression tree in the filter buffer.

11 Call NWDSSearch (page 383) to start the search. Make sure iterationHandle is set to NO_MORE_ITERATIONS before calling the function.

12 Call NWDSGetObjectCount (page 210) to determine the number of objects whose information is stored in the result buffer.

13 Call NWDSGetObjectName (page 214) to get the name of the current object in the buffer and the count of attributes associated with the object.

14 Call NWDSGetAttrName (page 173) to retrieve the name of the attribute and the count of values associated with the attribute.

15 Call NWDSComputeAttrValSize (page 130) to determine the size of the attribute value; then allocate a block of memory that size.

16 For each value associated with the attribute, call NWDSGetAttrVal (page 175) to retrieve the value.

17 Free the memory that was allocated in Step15.

18 Loop to Step 14 until all attribute information for the object has been read.

19 Loop to Step 13 until the information for all objects in the buffer has been retrieved.

20 If the iteration handle is not equal to NO_MORE_ITERATIONS, loop to Step 11.

21 Call NWDSFreeBuf (page 158) to free the filter buffer.

22 Call NWDSFreeBuf (page 158) to free the request buffer.

23 Call NWDSFreeBuf (page 158) to free the result 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” on page 27.

See Also:

- ndssearc.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- Section 1.4, “Search Requests,” on page 30

2.5 Partition and Replica Tasks

This section provides procedures for reading information about partitions and replicas and for managing them (adding, moving, joining, splitting, and deleting).

2.5.1 Adding a Replica

To add a replica of an existing partition to a server, complete the following step.

1 Call NWDSAddReplica (page 91) with the name of the partition's root object.

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

See Also:

- “Changing the Type of a Replica” on page 65

2.5.2 Changing the Type of a Replica

To change a replica’s type, complete the following step.

1 Call NWDSChangeReplicaType (page 122).

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.

See Also:

- “Removing Replicas” on page 66

2.5.3 Joining Partitions

To join a subordinate partition to its parent, complete the following step.

1 Call NWDSJoinPartitions (page 246).

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

See Also:

- "Splitting Partitions" on page 67
- "Removing Partitions" on page 66
- "Changing the Type of a Replica" on page 65

2.5.4 Listing Partitions and Retrieving Partition Information

To obtain information about the partitions stored on a specified server, complete the following steps.

1. Allocate a result buffer to receive the results by calling NWDSAllocBuf (page 95). (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 (page 264).
4. Determine the number of partitions whose information is stored in the result buffer by calling NWDSGetServerName (page 234).
5. For each partition whose information is stored in the buffer, retrieve the partition name and type by calling NWDSGetPartitionInfo (page 224).
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 (page 158).

If you decide to stop retrieving partition information before iterationHandle is set to NO_MORE_ITERATIONS, call NWDSCloseIteration (page 126) to free memory and state information associated with the partition listing operation.

See Also:

- "Joining Partitions" on page 65

2.5.5 Removing Partitions

To delete a partition, complete the following step.

1. Call NWDSRemovePartition (page 361).

   You can remove a partition only if there are no secondary or read-only replicas of the partition. The partition must also be empty, containing no other objects except the root container object.

See Also:

- "Adding a Replica" on page 65

2.5.6 Removing Replicas

To delete a replica of a partition, complete the following step.

1. Call NWDSRemoveReplica (page 363).

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: NWDSJoinPartitions removes the master replica of the subordinate partition.

See Also:

- "Adding a Replica" on page 65
- "Changing the Type of a Replica" on page 65

2.5.7 Splitting Partitions

To divide a partition at a specified object, complete the following step.

1 Call NWDSSplitPartition (page 394).

The specified object becomes the root object of the subordinate partition. Split operations are always performed on the master replica.

See Also:

- "Removing Partitions" on page 66

2.6 Schema Tasks

This section provides procedures for modifying the schema by deleting or creating classes and attributes. It also provides procedures for reading the schema definitions: classes, attributes, and syntaxes. See one of the following:

- Section 2.6.1, "Creating a Class Definition," on page 67
- Section 2.6.2, "Creating an Attribute Definition," on page 68
- Section 2.6.3, "Deleting a Class Definition," on page 69
- Section 2.6.4, "Deleting an Attribute Definition," on page 69
- Section 2.6.5, "Listing Containable Classes," on page 69
- Section 2.6.6, "Modifying a Class Definition," on page 70
- Section 2.6.7, "Reading a Class Definition," on page 70
- Section 2.6.8, "Reading an Attribute Definition," on page 72
- Section 2.6.9, "Retrieving Syntax Names and Definitions," on page 73

2.6.1 Creating a Class Definition

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

1 Allocate a structure of type Class_Info_T (page 439) and fill it in.

2 Allocate a request buffer by calling NWDSAllocBuf (page 95).

3 Initialize the request buffer for a DSV_DEFINE_CLASS operation by calling NWDSInitBuf (page 242).

4 Prepare the request buffer for storing object-class names in the Super Class Names list by calling NWDSBeginClassItem (page 110).

5 Place the desired object-class names in the Super Class List by calling NWDSPutClassItem (page 319) once for each object-class name to be placed in the list.

6 Prepare the request buffer for storing object-class names in the Containment Class Names list by calling NWDSBeginClassItem (page 110).

7 Place the desired object-class names in the Containment Class Names list by calling NWDSPutClassItem (page 319) once for each object-class name to be placed in the list.

8 Prepare the request buffer for storing naming-attribute names in the Naming Attributes List by calling NWDSBeginClassItem (page 110).

9 Place the desired naming-attribute names in the Naming Attributes List by calling NWDSPutClassItem (page 319) once for each naming-attribute name to be placed in the list.

10 Prepare the request buffer for storing attribute names in the Mandatory Attribute Names list by calling NWDSBeginClassItem (page 110).

11 Place the desired attribute names in the Mandatory Attribute Names list by calling NWDSPutClassItem (page 319) once for each attribute name to be placed in the list.

12 Prepare the request buffer for storing attribute names in the Optional Attribute Names list by calling NWDSBeginClassItem (page 110).

13 Place the desired attribute names in the Optional Attribute Names list by calling NWDSPutClassItem (page 319) once for each attribute name to be added to the list.

14 Add the object-class definition to the eDirectory Schema by calling NWDSDefineClass (page 137).

15 Free the request buffer by calling NWDSFreeBuf (page 158).

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 (page 110) to move to the list. You then immediately call NWDSBeginClassItem (page 110) again to move to the next list.

See Also:

- crclsdef.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- "Valid Class and Attribute Names"

2.6.2 Creating an Attribute Definition

1 Declare a structure of type Attr_Info_T (page 433) and fill it.

2 Create a new attribute definition to define the attribute by calling NWDSDefineAttr (page 135).

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.

See Also:

- "Reading an Attribute Definition" on page 72

2.6.3 Deleting a Class Definition

You can delete a class definition using a straightforward function called NWDSRemoveClassDef (page 357), 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 eDirectory tree.

See Also:

- "Listing Containable Classes" on page 69

2.6.4 Deleting an Attribute Definition

Delete an attribute definition by calling NWDSRemoveAttrDef (page 355). 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 requires only 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.

See Also:

- "Creating a Class Definition" on page 67

2.6.5 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 (page 95). (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 (page 258).
4. Determine the number of object-class names contained in the buffer by calling NWDSGetClassItemCount (page 189).
5. For each object class name in the result buffer, retrieve the name by calling NWDSGetClassItem (page 187).

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 (page 158).

Aborting Iterative Operations

If you need information on aborting an iterative operation, see "Controlling Iterations" on page 27.

See Also:

- "Retrieving Syntax Names and Definitions" on page 73

2.6.6 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 (page 95).

2 Initialize the request buffer for a DS_MODIFY_CLASS_DEF operation by calling NWDSInitBuf (page 242).

3 For each optional attribute to be added to the class definition, store the attribute's name in the request buffer by calling NWDSPutAttrName (page 308).

4 Modify the object class definition by calling NWDSModifyClassDef (page 281).

5 Free the request buffer by calling NWDSFreeBuf (page 158).

See Also:

- "Reading a Class Definition" on page 70

2.6.7 Reading a Class Definition

Use the following steps to retrieve information about object-class definitions in the eDirectory Schema.

1 Allocate a request buffer by calling NWDSAllocBuf (page 95). (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 (page 242).

3 For each object class you want to receive information about, store the object-class name in the request buffer by calling NWDSPutClassName (page 321).

4 Allocate a result buffer by calling NWDSAllocBuf (page 95). (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 (page 333).

7 Determine the number of object classes whose information is in the result buffer by calling NWDSGetClassDefCount (page 185).

8 If you have chosen object names and definitions, retrieve each object-class definition from the buffer by using the steps in the "Object Class Procedure List" that follows 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 (page 158).

11 Free the result buffer by calling NWDSFreeBuf (page 158).

Object Class Procedure List. 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 (page 183).

   If you called NWDSReadClassDef (page 333) with infoType set to DS_INFO_CLASS_DEFS, skip the rest of the following steps. With this infoType, the NWDSGetClassDef function retrieves all the information about the class from the buffer. Repeat Step 1 for each object-class definition in the buffer.

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 (page 189). (Moving to the list and determining the number of class names is accomplished with one call to NWDSGetClassItemCount (page 189).)

3 For each super-class name in the Super Class Names list, retrieve the super-class name by calling NWDSGetClassItem (page 187).

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 (page 189).

5 For each containment class name in the Containment Class Names list, retrieve the containment-class name by calling NWDSGetClassItem (page 187).

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 (page 189).

7 For each naming-attribute name in the Naming Attribute Names list, retrieve the naming-attribute name by calling NWDSGetClassItem (page 187).

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 (page 189).

9 For each mandatory-attribute name in the Mandatory Attribute Names list, retrieve the naming attribute name by calling NWDSGetClassItem (page 187).

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 (page 189).

11 For each optional-attribute name in the Optional Attribute Names list, retrieve the optional-attribute name by calling NWDSGetClassItem (page 187).

If you want to determine the names of all of the classes and their definitions, use NWDSReadClassDef (page 333) 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);
```

NOTE: A call to NWDSReadClassDef (page 333) 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.

See Also:

- rdclsdef.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- crclsdef.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

2.6.8 Reading an Attribute Definition

To retrieve information about selected attribute definitions use the following steps:

1. Allocate a request buffer by calling NWDSAllocBuf (page 95). (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 (page 242).
3. For each attribute whose information you want to retrieve, store the attribute's name in the request buffer by calling NWDSPutAttrName (page 308).
4. Allocate a result buffer by calling NWDSAllocBuf (page 95). (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 NWDSReadAttrDef (page 330) with infoType set to DS_ATTR_DEF_NAMES to retrieve names only, or set to 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 (page 169).
8. For each attribute in the buffer, remove the attribute information by calling NWDSGetAttrDef (page 171).
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 (page 158).
11. Free the result buffer by calling NWDSFreeBuf (page 158).

Reading an attribute definition by using NWDSReadAttrDef (page 330) 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);
```

See Also:

- rdattdef.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)
- readinfo.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm)

2.6.9 Retrieving Syntax Names and Definitions

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

1. Allocate memory for the result buffer by calling NWDSAllocBuf (page 95). (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 (page 348) with infoType =DS_SYNTAX_DEFS, allSyntaxes =TRUE, and syntaxNames =NULL.
4. Determine the number of syntax definitions in the result buffer by calling NWDSGetSyntaxCount (page 236).
5. For each syntax in the result buffer, retrieve the syntax name and definition by calling NWDSGetSyntaxDef (page 238).
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 (page 158).

To retrieve information about specific syntaxes, rather than all syntaxes in the eDirectory Schema, do the following:

1. Allocate memory for the request buffer by calling NWDSAllocBuf (page 95).
2. Initialize the request buffer for a DSV_READ_SYNTAXES operation by calling NWDSInitBuf (page 242).
3. For each syntax whose information you want to retrieve, store the syntax's name in the request buffer by calling NWDSPutSyntaxName (page 325).
4. Allocate memory for the result buffer by calling NWDSAllocBuf (page 95). (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 (page 348) 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 (page 236).
8. For each syntax in the buffer, retrieve the syntax name and definition by calling NWDSGetSyntaxDef (page 238).
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 (page 158).
11. Free the result buffer by calling NWDSFreeBuf (page 158).

See Also:

- "Creating an Attribute Definition" on page 68

Functions

3

The following eDirectory services have been separated out into separate books:

- *NDK: eDirectory Event Services*
- *NDK: Novell eDirectory Iterator Services*
- *NDK: eDirectory Backup Services*

IMPORTANT: All the NWDS functions pass in a context handle parameter. This parameter affects whether the function uses distinguished names or relative distinguished names. For more information, see Section 1.1, “Context Handles,” on page 15.

This chapter lists alphabetically the core Novell® eDirectory™ functions and describes their purpose, syntax, parameters, and return codes.

NWDSAbbreviateName

Converts an NDS name (including the naming attributes) to its shortest form relative to a specified name context.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsname.h>
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAbbreviateName  (
   NWDSContextHandle   context,
   pnstr8              inName,
   pnstr8              abbreviatedName);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 Unicode characters (Unicode characters are always 16 bits). 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
```

with a context of 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 (page 114)

NWDSAbortPartitionOperation

Aborts a partition operation in progress.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAbortPartitionOperation  (
   NWDSContextHandle   context,
   pnstr8              partitionRoot);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSJoinPartitions (page 246), NWDSSplitPartition (page 394)

NWDSAddFilterToken

Adds a node to the search filter expression tree.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAddFilterToken  (
   pFilter_Cursor_T   cur,
   nuint16            tok,
   nptr               val,
   nuint32            syntax);
```

Pascal

```
uses netwin32

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 (see Section 5.13, “Filter Tokens,” on page 474).

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 (see Section 5.26, "Syntax IDs," on page 487).

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

Each node contains a token and a syntax with an associated value. 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 create a search filter and for more details about eDirectory searches, see Section 1.4, "Search Requests," on page 30. For step-by-step instructions, see "Searching eDirectory" on page 63. For sample code, see ndssearc.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm).

NCP Calls

None

See Also

NWDSAllocFilter (page 97), NWDSDelFilterToken (page 140), NWDSFreeFilter (page 162), NWDSPutFilter (page 323)

NWDSAddObject

Adds an object to the eDirectory tree.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAddObject  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnint_ptr           iterationHandle,
   nbool8              more,
   pBuf_T              objectInfo);
```

Pascal

```
uses netwin32

Function NWDSAddObject
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint_ptr;
   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. This should be set to NO_MORE_ITERATIONS initially.

more

(IN) Specifies whether additional information will be returned:

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 value.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

For NWDSAddObject to succeed, the new object’s immediate superior must already exist.

For information on setting the NDS context for the context parameter, see Section 1.1, “Context Handles,” on page 15.

The objectName parameter 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 eDirectory ensures that new objects conform to the eDirectory 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 Section 1.6, "Add Object Requests," on page 38.

For step-by-step instructions, see "Adding an eDirectory Object" on page 58.

For sample code, see ndsadd1.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm).

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 eDirectory NCP

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSListContainableClasses (page 258), NWDSModifyObject (page 286), NWDSRemoveObject (page 359)

NWDSAddPartition (obsolete—moved from .h file 11/99)

Was last documented in September 1999. Call NWDSAddReplica (page 91) and NWDSSplitPartition (page 394) instead.

NWDSAddReplica

Adds a replica of an existing eDirectory partition to a server.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAddReplica  (
   NWDSContextHandle   context,
   pnstr8              server,
   pnstr8              partitionRoot ;
   nuint32             replicaType);
```

Pascal

```
uses netwin32

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 eDirectory partition to be replicated.

replicaType

(IN) Specifies the type of the new replica (see Section 5.23, “Replica Types,” on page 483).

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The partition must be created beforehand by calling NWDSSplitPartition.

NOTE: You cannot create a master replica type (RT_MASTER) with NWDSAddReplica. To make a new replica the mater replica, use NWDSChangeReplicaType.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSChangeReplicaType (page 122), NWDSRemoveReplica (page 363), NWDSSplitPartition (page 394)

NWDSAddSecurityEquiv

Adds to the specified object's security equivalence.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAddSecurityEquiv  (
   NWDSContextHandle   context,
   pnstr8              equalFrom,
   pnstr8              equalTo);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRemSecurityEquiv (page 365)

NWDSAllocBuf

Allocates a Buf_T structure for use as a request or result buffer parameter to an eDirectory function.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAllocBuf  (
   size_t    size,
   ppBuf_T   buf);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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)+3 which should be less than 64K bytes.

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 (page 158), NWDSInitBuf (page 242)

NWDSAllocFilter

Allocates a filter expression tree and initializes a cursor to the current insertion point.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAllocFilter  (
   ppFilter_Cursor_T    cur);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Search tokens are added to the filter with calls to NWDSAddFilterToken.

NCP Calls

None

See Also

NWDSAddFilterToken (page 84), NWDSDelFilterToken (page 140), NWDSFreeFilter (page 162), NWDSPutFilter (page 323)

NWDSAuditGetObjectID (obsolete 06/03)

Returns a connection handle and an object ID for the object name, but is now obsolete.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsaud.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAuditGetObjectID  (
   NWDSContextHandle      context,
   pnstr8                 objectName,
   NWCONN_HANDLE N_FAR   *conn,
   pnuint32               objectID);
```

Pascal

```
uses netwin32

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 eDirectory object ID.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The returned connection handle is the NetWare server where the object is stored.

Use NWDSResolveName in place of this function.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAddObject (page 87), NWDSResolveName (page 371)

NWDSAuthenticate (obsolete 06/03)

Establishes an authenticated connection to a secured NetWare server using the unauthenticated connection and local data cached by calling NWDSLogin, but is now obsolete.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSAuthenticate  (
   NWCONN_HANDLE         conn,
   nflag32               optionsFlag,
   pNWDS_Session_Key_T   sessionKey);
```

Pascal

```
uses netwin32

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) Specifies reserved; pass in a zero (0).

sessionKey

(IN) Points to reserved; pass in NULL.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSAuthenticate first checks to see if the specified connection is authenticated. If the connection is authenticated, NWDSAuthenticate will return SUCCESSFUL and end the call.

Use NWDSAuthenticateConn or NWDSAuthenticateConnEx in place of this function.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAuthenticateConn (page 103), NWDSLogin (page 270)

NWDSAuthenticateConn

Authenticates and licenses an established connection to a NetWare server.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSAuthenticateConn  (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle);
```

Pascal

```
uses netwin32

Function NWDSAuthenticateConn
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(IN) Specifies the connection handle to authenticate.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSAuthenticateConn authenticates and licenses the connection using the identity established by calling NWDSLogin. An authenticated, licensed connection has access to eDirectory and the file system.

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.

If the specified connection is already authenticated, the function returns SUCCESSFUL and ends the call.

NCP Calls

0x2222 104 2 Send eDirectory Fragmented Request/Reply

See Also

NWDSOpenConnToNDSServer (page 297), NWCCOpenConnByName (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk645.html)

NWDSAuthenticateConnEx

Authenticates, but does not license, an established connection to a NetWare server.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSAuthenticateConnEx  (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle);
```

Pascal

```
uses netwin32

Function NWDSAuthenticateConnEx
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(IN) Specifies the connection handle to authenticate.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSAuthenticateConnEx authenticates the connection using the identity, established by calling NWDSLogin, of the object derived from the context handle. The connection is authenticated not licensed. Such connections have access to eDirectory; to access the file system the connection must be licensed.

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.

If the specified connection is already authenticated, the function returns SUCCESSFUL and ends the call.

To license an authenticated connection, call NWCCLicenseConn (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk625.html).

NCP Calls

0x2222 104 2 Send eDirectory Fragmented Request/Reply

See Also

NWDSOpenConnToNDSServer (page 297), NWDSOpenMonitoredConn (page 299), NWCCOpenConnByName (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk645.html), NWCCLicenseConn (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk625.html)

NWDSBackupObject

Backs up the attribute names and values for an object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSBackupObject  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnint_ptr           iterationHandle,
   pBuf_T              objectInfo);
```

Pascal

```
uses netwin32

Function NWDSBackupObject
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint_ptr;
   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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSBackupObject is used to back up the attributes and attribute values for one object at a time. To back up eDirectory, 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.

To abort the operation before retrieving all the information about an object, call NWDSCloseIteration with an operation type of DSV_BACKUP_ENTRY.

IMPORTANT: 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 eDirectory 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRestoreObject (page 373)

NWDSBeginClassItem

Begins a class item definition (which is a part of an object class definition) in a request buffer to be used by a eDirectory Schema function.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSBeginClassItem  (
   NWDSContextHandle   context,
   pBuf_T              buf);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The buf parameter 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 sName 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 (page 321), NWDSPutClassItem (page 319)

NWDSCanDSAuthenticate

Determines if eDirectory credentials exist for the specified tree name.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSCanDSAuthenticate  (
   NWDSContextHandle   context);
```

Pascal

```
uses netwin32

Function NWDSCanDSAuthenticate
  (context : NWDSContextHandle
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

Return Values

These are common return values.

0	No credentials found
1	Credentials exist

Remarks

NWDSCanDSAuthenticate is similar to NWIsDSAuthenticated, but is enabled for multiple tree environments. NWDSCanDSAuthenticate indicates if eDirectory 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 (page 103), NWIsDSAuthenticated (page 419)

NWDSCanonicalizeName

Converts an abbreviated name to the canonical form.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsname.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSCanonicalizeName  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              canonName);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 value in the context parameter is

```
OU=Looney Toons.O=Acme
```

the canonicalized name is

```
CN=Elmer Fudd.OU=Looney Toons.O=Acme
```

The objectName parameter supplies the abbreviated form of a eDirectory name. The name can be typed or typeless. If a typeless name is supplied, the NWDSCanonicalizeName function uses default typing rules to assign types. The wrong types will be assigned if a component of a typeless name is a Country, Locality, Tree Root, or domain. For more information about the typing rules and a method that ensures accurate results, see the DCV_TYPELESS_NAMES key in “DCK_FLAGS Key” on page 18.

The name can also be truncated. It is assumed that a truncated name is relative to the naming path supplied by the specified context.

The canonName parameter 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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

NCP Calls

None

See Also

NWDSAbbreviateName (page 80)

NWDSChangeObjectPassword

Changes the authentication password for an eDirectory object once a public/private key pair has been assigned. Does not support international or extended characters in passwords.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSChangeObjectPassword  (
   NWDSContextHandle   context,
   nflag32             pwdOption,
   pnstr8              objectName,
   pnstr8              oldPassword,
   pnstr8              newPassword);
```

Pascal

```
uses netwin32

Function NWDSChangeObjectPassword
  (context : NWDSContextHandle;
   pwdOption : nflag32;
   objectName : pnstr8;
   oldPassword : pnstr8;
   newPassword : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

pwdOption

(IN) Specifies which password to change. Select from the following:

Value	Constant	Description
0	ALL_PASSWORDS	All passwords are changed.
1	NDS_PASSWORD	Only the eDirectory password is changed.
2	NT_PASSWORD	Only the NT password is changed (the NT password that NDS4NT in eDirectory).
4	AD_PASSWORD	Only the AD/NT password (serviced by password sync).

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values".

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.

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.

The NT_PASSWORD option works only if the eDirectory for NT product has been installed. If this flag is passed in and the eDirectory for NT product has not been installed, an error is returned.

If the ALL_PASSWORDS option is set, the NDS_PASSWORD operation is performed first. If successful, other password operations are attempted, but error conditions are not returned for the other operations.

NOTE: NWDSChangePwdEx (page 119) supports international and extended characters and is recommended in place of NWDSChangeObjectPassword.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSGenerateObjectKeyPair (page 167)

NWDSChangePwdEx

Changes the authentication password for an eDirectory object once a public/private key pair has been assigned. Supports international and extended characters in passwords.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSChangePwdEx  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   nuint32             pwdFormat,
   nptr                oldPwd,
   nptr                newPwd,
   nuint32             pwdOption);
```

Pascal

```
uses netwin32

Function NWDSChangePwdEx
  (context : NWDSContextHandle;
   objectName : pnstr8;
   pwdFormat : nuint32;
   oldPwd : nptr;
   newPwd : nptr;
   pwdOption : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

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

pwdFormat

(IN) Specifies the format of the password data. Select from the following:

PWD_UNICODE_STRING
PWD_UTF8_STRING
PWD_RAW_C_STRING

oldPwd

(IN) Points to the object's current password.

newPwd

(IN) Points to the object's new password.

pwdOption

(IN) Specifies which password to change. Select from the following:

Value	Constant	Description
0	ALL_PASSWORDS	All passwords are changed.
1	NDS_PASSWORD	Only the eDirectory password is changed.
2	NT_PASSWORD	Only the NT password is changed (the NT password that NDS4NT in eDirectory).
4	AD_PASSWORD	Only the AD/NT password (serviced by password sync).

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values".

Remarks

For NWDSChangePwdEx to succeed, oldPwd 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, newPwd 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.

The NT_PASSWORD option works only if the eDirectory for NT product has been installed. If this option is set in and the eDirectory for NT product has not been installed, an error is returned.

If the ALL_PASSWORDS option is set, the NDS_PASSWORD operation is performed first. If successful, other password operations are attempted, but error conditions are not returned for the other operations.

NOTE: The PWD_RAW_C_STRING password format allows any arbitrary NULL-terminiated data to be used as a password. Passwords specified with this format are not interoperable with unicode and UTF8 passwords.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSGenerateKeyPairEx (page 164)

NWDSChangeReplicaType

Changes the replica type of a given replica on a given server.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSChangeReplicaType  (
   NWDSContextHandle   context,
   pnstr8              replicaName,
   pnstr8              server,
   nuint32             newReplicaType);
```

Pascal

```
uses netwin32

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 eDirectory partition whose replica type will be changed.

server

(IN) Points to the name of the server on which the replica resides.

newReplicaType

(IN) Specifies the replica type the given replica is to be changed to (see Section 5.23, "Replica Types," on page 483).

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

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 change only 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSSplitPartition (page 394), NWDSJoinPartitions (page 246), NWDSAddReplica (page 91), NWDSRemoveReplica (page 363)

NWDSCIStringsMatch

Tests two case ignore strings (defined by CI_String_T) to determine if the two strings are equivalent.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsname.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSCIStringsMatch  (
   NWDSContextHandle   context,
   pnstr8              string1,
   pnstr8              string2,
   pnint               matches ;)
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory attributes such as CN, Description, Given Name, Surname, and Title.

Depending on the setting of the DCV_XLATE_STRINGS context key, NWDSCIStringsMatch compares two strings either 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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>
#include <nwdsdefs.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSCloseIteration  (
   NWDSContextHandle   context,
   nint32              iterationHandle,
   nuint32             operation);
```

Pascal

```
uses netwin32

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 eDirectory operation associated with iterationHandle (see Section 5.3, "Buffer Operation Types and Related Functions," on page 464).

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRead (page 327), NWDSList (page 248), NWDSSearch (page 383), NWDSListAttrsEffectiveRights (page 251), NWDSBackupObject (page 107), NWDSRestoreObject (page 373), NWDSListPartitions (page 264), NWDSListContainableClasses (page 258), NWDSReadAttrDef (page 330), NWDSReadClassDef (page 333)

NWDSCompare

Compares an object's attribute value with a specified value.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSCompare  (
   NWDSContextHandle   context,
   pnstr8              object,
   pBuf_T              buf,
   pnbool8             matched);
```

Pascal

```
uses netwin32

Function NWDSCompare
  (context : NWDSContextHandle;
   objectName : pnstr8;
   buf : pBuf_T;
   matched : pnbool8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

object

(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 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

The input buffer, buf, should be allocated with the NWDSAllocBuf function and initialized for the DSV_COMPARE operation with the NWDSInitBuf function.

The matched parameter receives a Boolean indicating the result of the comparison. The result is TRUE if the comparison was successful; otherwise, the result is FALSE.

For step-by-step instructions, see “Comparing Attribute Values” on page 58.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRead (page 327)

NWDSComputeAttrValSize

Computes, in conjunction with NWDSGetAttrVal, the size of the attribute value at the current position in the result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSComputeAttrValSize  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   nuint32             syntaxID,
   pnuint32            attrValSize);
```

Pascal

```
uses netwin32

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 (see Section 5.26, “Syntax IDs,” on page 487).

attrValSize

(OUT) Points to the size (in bytes) required to retrieve the attribute.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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.

If NWDSRead is called using infoTypes DS_VALUE_INFO (3) or DS_ABBREVIATED_VALUE (4), eDirectory returns the attribute value flags and modification timestamp. If the attribute value has been deleted, but not synchronized, eDirectory can return a value flag of DS_NOT_PRESENT (value not present) and no data. In this case, the attribute value length is zero. NWDSComputeAttrValSize returns the correct size of 0, and the application should not call NWDSGetAttrVal to retrieve the data because there is no data to retrieve.

Call NWDSComputeAttrValSize once for each attribute value you retrieve from the result buffer.

The syntaxID parameter identifies the syntax data type the attribute information is stored in. The data structures associated with the syntaxes are listed in "Attribute Syntax Definitions". The enumerated types for syntaxes (such as SYN_DIST_NAME) are located in NWDSDEFS.H. The NWDSGetAttrName function returns the syntax ID of the attribute.

The attrValSize parameter 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.

For complete steps on reading the information from the buffer, see "Reading Attributes of eDirectory Objects" on page 62.

NCP Calls

None

See Also

NWDSGetAttrVal (page 175)

NWDSCreateContext (obsolete—moved from .h file 6/99)

Was last documented in June 1999. Call NWDSCreateContextHandle (page 133) instead.

NWDSCreateContextHandle

Creates a new context handle and initializes it with default values.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSCreateContextHandle  (
   NWDSContextHandle N_FAR   *newHandle);
```

Pascal

```
uses netwin32

Function NWDSCreateContextHandle
  (Var newHandle : NWDSContextHandle
) : NWDSCCODE;
```

Parameters

newHandle

(OUT) Points to the newly created context handle.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

NWDSCreateContextHandle allocates a new context handle and initializes the context handle with default values (see Section 5.7, "Default Context Key Values," on page 469).

The number of context handles an application can create is limited only by available resources. Creation can fail if there is insufficient memory or the Unicode tables have not been initialized (see NWInitUnicodeTables (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/

ucod_enu/data/sdk143.html) (*Unicode* (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/ucod_enu/data/hjg275fp.html))).

To view or modify context information, use the NWDSGetContext (page 191) and the NWDSSetContext (page 387) functions.

NCP Calls

None

See Also

NWDSDuplicateContextHandle (page 144), NWDSFreeContext (page 160), NWDSGetContext (page 191), NWDSSetContext (page 387)

NWDSDefineAttr

Adds a new attribute definition to the eDirectory schema.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdssch.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSDefineAttr  (
   NWDSContextHandle   context,
   pnstr8              attrName,
   pAttr_Info_T        attrDef);
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

The name of the new attribute must be unique within the eDirectory schema attribute definitions. The names of the attributes for the base schema are listed in "Base Attribute Definitions" (*NDK: Novell eDirectory Schema Reference*). 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 Novell 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSDefineClass (page 137)

NWDSDefineClass

Adds a new object class definition to the eDirectory schema.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdssch.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSDefineClass  (
   NWDSContextHandle   context,
   pnstr8              className,
   pClass_Info_T       classInfo,
   pBuf_T              classItems);
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

The name of the new object class must be unique within the eDirectory schema class definitions. The names of the classes for the base schema are listed in "Base 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 Novell Developer Support to guarantee uniqueness.

The classItems parameter 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

For step-by-step instructions, see "Creating a Class Definition" on page 67.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSDefineAttr (page 135), NWDSModifyClassDef (page 281)

"Valid Class and Attribute Names" (*NDK: Novell eDirectory Schema Reference*)

NWDSDelFilterToken

Deletes the most recently added token from a filter expression tree.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSDelFilterToken  (
   pFilter_Cursor_T   cur,
   void               (N_FAR N_CDECL   *freeVal)(
                              nuint32   syntax,
                              nptr      val));
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The freeVal parameter 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.The freeVal parameter 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).

For syntax IDs (such as SYN_BOOLEAN), see Section 5.26, “Syntax IDs,” on page 487.

NCP Calls

None

See Also

NWDSAddFilterToken (page 84), NWDSAllocFilter (page 97), NWDSFreeFilter (page 162), NWDSPutFilter (page 323)

NWDSDuplicateContext (obsolete 03/99)

Creates an NDS context and initializes it to the same settings as an existing NDS context. This function is obsolete. Call NWDSDuplicateContextHandle (page 144) instead.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSDuplicateContext  (
   NWDSContextHandle   oldContext);
```

Pascal

```
uses netwin32

Function NWDSDuplicateContext
  (oldContext : NWDSContextHandle
) : NWDSContextHandle;
```

Parameters

oldContext

(IN) Specifies the NDS context to duplicate.

Return Values

These are common return values; see "NDS Return Values" for more information.

0x0000 0000	SUCCESSFUL
0xFFFF FEB8	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 (page 133), NWDSFreeContext (page 160), NWDSSetContext (page 387), NWDSGetContext (page 191)

NWDSDuplicateContextHandle

Allocates memory for a new context structure and initializes it with values copied from the source context structure.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdc.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSDuplicateContextHandle  (
   NWDSContextHandle           srcContextHandle,
   NWDSContextHandle N_FAR    *destContextHandle);
```

Pascal

```
uses netwin32

Function NWDSDuplicateContextHandle
  (srcContextHandle : NWDSContextHandle;
   Var destContextHandle : 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.

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

NCP Calls

None

See Also

NWDSCreateContextHandle (page 133), NWDSDuplicateContext (obsolete 03/99) (page 142)

NWDSExtSyncList

Lists the immediate subordinates for an eDirectory object and places restrictions on the subordinate's names, classes, modification times, and object types.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSExtSyncList  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              className,
   pnstr8              subordinateName,
   pnint_ptr           iterationHandle,
   pTimeStamp_T        timeStamp,
   nbool               onlyContainers,
   pBuf_T              subordinates);
```

Pascal

```
uses netwin32

Function NWDSExtSyncList
  (context : NWDSContextHandle;
   objectName : pnstr8;
   className : pnstr8;
   subordinateName : pnstr8;
   iterationHandle : pnint_ptr;
   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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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."
```

The timeStamp 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.

The iterationHandle parameter controls retrieval of 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 the result buffer holds the complete results when NWDSExtSyncList returns from its initial call, 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.

The onlyContainers parameter 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 eDirectory Output Buffers” on page 53.

NOTE: On large networks, iterative processes, such as NWDSExtSyncList, might take a long 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSList (page 248), NWDSListByClassAndName (page 254), NWDSListContainers (page 261)

NWDSExtSyncRead

Reads values from one or more of an eDirectory object's attributes and places restrictions on the attributes' modification time.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSExtSyncRead  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint_ptr           iterationHandle,
   pTimeStamp_T        timeStamp,
   pBuf_T              objectInfo);
```

Pascal

```
uses netwin32

Function NWDSExtSyncRead
  (context : NWDSContextHandle;
   objectName : pnstr8;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint_ptr;
   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 (see Section 5.16, “Information Types for Search and Read,” on page 476).

allAttrs

(IN) Specifies the scope of the request: TRUE=information concerning all attributes is requested; FALSE=only attributes named in the attrNames parameter are requested.

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The name specified by objectName is relative to the current name context in the NDS context specified by context.

The infoType, allAttrs, attrNames, and timeStamp parameters indicate what attribute information is requested.

The infoType specifies whether both attribute names and attribute values are requested

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, and 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 to be returned. Initialize the buffer with a value of DSV_READ. For more information on setting up this buffer, see “Reading Attributes of eDirectory Objects” on page 62.

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 eDirectory Output Buffers” on page 53.

If the infoType parameter is set to 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.

The iterationHandle parameter controls retrieval of search results larger than the result buffer pointed to by objectInfo.

Before the initial call to NWDSExtSyncRead, set the iterationHandle parameter to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when NWDSExtSyncRead 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 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 eDirectory NCP

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRead (page 327), NWDSReadObjectInfo (page 340)

NWDSExtSyncSearch

Searches a region of the eDirectory tree for objects satisfying a set of specified requirements, including modification time.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSExtSyncSearch  (
   NWDSContextHandle   context,
   pnstr8              baseObjectName,
   nint                scope,
   nbool8              searchAliases,
   pBuf_T              filter
   pTimeStamp_T        timeStamp,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint_ptr           iterationHandle,
   nint32              countObjectsToSearch,
   pnint32             countObjectsSearched,
   pBuf_T              objectInfo);
```

Pascal

```
uses netwin32

Function NWDSExtSyncSearch
  (context : NWDSContextHandle;
   baseObjectName : pnstr8;
   scope : nint;
   searchAliases : nbool8;
   filter : pBuf_T;
   timeStamp : pTimeStamp_T;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
```

```
   iterationHandle : pnint_ptr;
   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 (see Section 5.22, “Scope Flags,” on page 483).

searchAliases

(IN) Specifies whether to dereference subordinate 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 (see Section 5.16, “Information Types for Search and Read,” on page 476).

allAttrs

(IN) Specifies the scope of the request: TRUE=information concerning all attributes is requested; FALSE=only attributes named in attrNames are 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 initially to NO_MORE_ITERATIONS.

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

NWDSExtSyncSearch succeeds if the base object specified by baseObjectName is located, regardless of whether there are any subordinates to return.

The baseObjectName parameter 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 of the base object are dereferenced while locating the base object unless the context flag associated with DCV_DEREF_ALIASES is not set.

The searchAliases parameter determines whether the aliases among the subordinates of the base object are dereferenced during the search. If TRUE, the search continues in the subtree of the aliases object. If FALSE, the search returns information about the alias object.

The filter parameter 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. For information about creating a filter, see Section 1.4, "Search Requests," on page 30. For step-by-step instructions, see "Searching eDirectory" on page 63.

When filling out TimeStamp_T, set eventID to zero, replicaNum to zero, and wholeSeconds to the appropriate value.

The infoType, allAttrs, and attrNames parameters 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, and 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 to be returned. Initialize this buffer with a value of DSV_SEARCH. For more information, see "Preparing eDirectory Input Buffers" on page 52.

On return, the buffer pointed to by objectInfo contains the information for objects matching the search criteria, along with the requested attribute information. This buffer is allocated by calling NWDSAllocBuf, but it is not initialize. For more information on reading the results, see "Retrieving Results from eDirectory Output Buffers" on page 53.

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

The iterationHandle parameter controls retrieval of search results larger than the result buffer pointed to by objectInfo.

Before the initial call to NWDSExtSyncSearch, set the iterationHandle parameter to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when NWDSExtSyncSearch 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 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.

NOTE: On large networks, iterative processes, such as NWDSExtSyncSearch, might take a long time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. Developers should use NWDSCloseIteration to allow users of their applications to abort an iterative process that is taking too long to complete.

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 are repeated in 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.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAddFilterToken (page 84), NWDSAllocFilter (page 97), NWDSCloseIteration (page 126), NWDSFreeFilter (page 162), NWDSPutFilter (page 323), NWDSSearch (page 383)

NWDSFreeBuf

Frees a buffer allocated by the NWDSAllocBuf function.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSFreeBuf  (
   pBuf_T    buf);
```

Pascal

```
uses netwin32

Function NWDSFreeBuf
  (buf : pBuf_T
) : NWDSCCODE;
```

Parameters

buf

(IN) Points to the buffer to be freed.

Return Values

These are common return values.

0x0000 0000	SUCCESSFUL
0xFFFF FEB5	ERR_NULL_POINTER
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 (page 95), NWDSInitBuf (page 242)

NWDSFreeContext

Frees a previously allocated NDS context.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSFreeContext  (
   NWDSContextHandle   context);
```

Pascal

```
uses netwin32

Function NWDSFreeContext
  (context : NWDSContextHandle
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context to be freed.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

All NDS contexts created by NWDSCreateContextHandle should be freed when the client is no longer using them. Doing so frees memory for the client.

NCP Calls

None

See Also

NWDSCreateContextHandle (page 133), NWDSGetContext (page 191), NWDSSetContext (page 387)

NWDSFreeFilter

Frees the area allocated to a search filter expression tree.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

N_EXTERN_LIBRARY (void)  NWDSFreeFilter  (
   pFilter_Cursor_T   cur,
   void               (N_FAR N_CDECL  *freeVal)(
                           nuint32   syntax,
                           nptr      val) );
```

Pascal

```
uses netwin32

Function NWDSFreeFilter
  (cur : pFilter_Cursor_T;
   freeVal : FreeValProc
);
```

Parameters

cur

(IN) Points to the filter to be freed, a filter previously allocated with NWDSAllocFilter.

freeVal

(IN) Specifies the function to be used to free nodes in the filter expression tree; can be NULL.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 (page 84), NWDSAllocFilter (page 97), NWDSDelFilterToken (page 140), NWDSPutFilter (page 323)

NWDSGenerateKeyPairEx

Creates or changes a public/private key pair for a specified object. Supports international and extended characters in passwords.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGenerateKeyPairEx  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   nuint32             pwdFormat,
   nptr                pwd,
   nuint32             pwdOption);
```

Pascal

```
uses netwin32

Function NWDSGenerateKeyPairEx
  (context : NWDSContextHandle;
   objectName : pnstr8;
   pwdFormat : nuint32;
   pwd : nptr;
   pwdOption : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

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

pwdFormat

(IN) Specifies the format of the password data. Select from the following:

PWD_UNICODE_STRING
PWD_UTF8_STRING
PWD_RAW_C_STRING

pwd

(IN) Points to the object password in the format specified by pwdFormat.

pwdOption

(IN) Specifies which password to operate on. Select from the following:

Value	Constant	Description
0	ALL_PASSWORDS	All passwords are changed.
1	NDS_PASSWORD	Only the eDirectory password is changed.
2	NT_PASSWORD	Only the NT password is changed (the NT password that NDS4NT in eDirectory).
4	AD_PASSWORD	Only the AD/NT password (serviced by password sync).

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values”.

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 object's password attributes before the NWDSGenerateObjectKeyPair function will succeed.

The NT_PASSWORD option only works if the eDirectory for NT product has been installed. If this option is set and the eDirectory for NT product has not been installed, an error is returned.

If the ALL_PASSWORDS option is set, the NDS_PASSWORD operation is performed first. If successful, other password operations are attempted, but error conditions are not returned for the other operations.

NOTE: The PWD_RAW_C_STRING password format allows any arbitrary NULL-terminiated data to be used as a password. Passwords specified with this format are not interoperable with unicode and UTF8 passwords.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSChangePwdEx (page 119)

NWDSGenerateObjectKeyPair

Creates or changes a public/private key pair for a specified object. Does not support international or extended characters in passwords.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGenerateObjectKeyPair  (
   NWDSContextHandle   contextHandle,
   pnstr8              objectName,
   pnstr8              objectPassword,
   nflag32             pwdOption);
```

Pascal

```
uses netwin32

Function NWDSGenerateObjectKeyPair
  (contextHandle : NWDSContextHandle;
   objectName : pnstr8;
   objectPassword : pnstr8;
   pwdOption : nflag32
) : NWDSCCODE;
```

Parameters

contextHandle

(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.

pwdOption

(IN) Specifies the password to operate on. Select from the following:

Value	Constant	Description
0	ALL_PASSWORDS	All passwords are changed.
1	NDS_PASSWORD	Only the eDirectory password is changed.
2	NT_PASSWORD	Only the NT password is changed (the NT password that NDS4NT in eDirectory).
4	AD_PASSWORD	Only the AD/NT password (serviced by password sync).

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values”.

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 object's password attributes before the NWDSGenerateObjectKeyPair function will succeed.

The NT_PASSWORD option only works if the eDirectory for NT product has been installed. If this flag is passed in as the value for the pwdOption parameter and the eDirectory for NT product has not been installed, an error is returned.

If the ALL_PASSWORDS option is set, the NDS_PASSWORD operation is performed first. If successful, other password operations are attempted, but error conditions are not returned for the other operations.

IMPORTANT: NWDSGenerateKeyPairEx (page 164) supports international and extended characters in passwords and is recommended in place of NWDSGenerateObjectKeyPair.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSChangeObjectPassword (page 116)

NWDSGetAttrCount

Returns the number of attributes whose information is stored in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetAttrCount  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            attrCount);
```

Pascal

```
uses netwin32

Function NWDSGetAttrCount
  (context : NWDSContextHandle;
   buf : pBuf_T;
   attrCount : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer being read.

attrCount

(OUT) Points to the number of attributes in the result buffer.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

The buf parameter points to a Buf_T filled in by a previous call to a eDirectory function, such as NWDSRead.

For complete steps on reading the information from the buffer, see “Reading Attributes of eDirectory Objects” on page 62.

NCP Calls

None

See Also

NWDSGetAttrDef (page 171), NWDSGetAttrName (page 173), NWDSRead (page 327), NWDSReadAttrDef (page 330)

NWDSGetAttrDef

Returns the next eDirectory Schema attribute definition from a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetAttrDef  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              attrName,
   pAttr_Info_T        attrInfo);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSGetAttrDef is used to retrieve attribute information from a result buffer filled in by NWDSReadAttrDef. For more information, see “Reading an Attribute Definition” on page 72.

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 Unicode characters (Unicode characters are always 16 bits). 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 (page 169), NWDSReadAttrDef (page 330)

NWDSGetAttrName

Retrieves the name of the attribute whose information is stored at the current position in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetAttrName  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              attrName,
   pnuint32            attrValCount,
   pnuint32            syntaxID);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
0xFFFF FEB5	ERR_NULL_POINTER
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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 Unicode characters (Unicode characters are always 16 bits). 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 still needs valid pointers for the attrValCount and syntaxID parameters, or the function returns ERR_NULL_POINTER.

For more information, see "Reading Attributes of eDirectory Objects" on page 62.

NCP Calls

None

See Also

NWDSGetAttrCount (page 169), NWDSRead (page 327), NWDSSearch (page 383), NWDSReadAttrDef (page 330)

NWDSGetAttrVal

Returns the next attribute value in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetAttrVal
  (NWDSContextHandle   context,
   pBuf_T              buf,
   nuint32             syntaxID,
   nptr                attrVal);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

NWDSGetAttrVal is used to retrieve attribute values from a result buffer filled in by functions such as NWDSList, NWDSRead, or NWDSSearch.

The syntaxID parameter is returned by a previous call to NWDSGetAttrName. The syntaxID parameter 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 "Attribute Syntax Definitions" (*NDK: Novell eDirectory Schema Reference*).

If the attrVal parameter 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.

For complete steps on reading the information from the buffer, see "Reading Attributes of eDirectory Objects" on page 62

NCP Calls

None

NWDSGetAttrValFlags

Returns the attribute value flags for the next attribute value in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API  NWDSGetAttrValFlags
  (NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            valueFlags);
```

Pascal

```
uses netwin32

Function NWDSGetAttrValFlags
  (context : NWDSContextHandle;
   buf : pBuf_T;
   valueFlags : pnuint32;
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer returned by NWDSRead when infoType is set to DS_VALUE_INFO.

valueFlags

(OUT) Points to the attribute value flags (see Section 5.2, “Attribute Value Flags,” on page 463).

Return Values

0x0000 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSGetAttrValFlags is used to retrieve attribute value flags from a result buffer filled in by functions such as NWDSList, NWDSRead, or NWDSSearch.

When NWDSRead is called with infoType equal to DS_VALUE_INFO, attribute names, values, value flags, and modification timestamps are returned in a result buffer. To retrieve information from the result buffer, the following sequence of calls should be made:

NWDSGetAttrValFlags
NWDSGetAttrValModTime
NWDSGetAttrVal

NCP Calls

None

NWDSGetAttrValModTime

Returns the modification timestamp for the next attribute value in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

NWDSCCODE N_API  NWDSGetAttrValModTime
  (NWDSContextHandle   context,
   pBuf_T              buf,
   pTimeStamp_T        timeStamp);
```

Pascal

```
uses netwin32

Function NWDSGetAttrValModTime
  (context : NWDSContextHandle;
   buf : pBuf_T;
   timeStamp : pTimeStamp_T;
 ) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

buf

(IN) Points to the result buffer returned from NWDSRead when infoType is set to DS_VALUE_INFO.

timeStamp

(OUT) Points to the modification timestamp of the attribute value.

Return Values

0x0000 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSGetAttrValModTime is used to retrieve the modification timestamp of attribute values from a result buffer filled in by functions such as NWDSList, NWDSRead, or NWDSSearch.

When NWDSRead is called with infoType equal to DS_VALUE_INFO, attribute names, values, value flags, and modification timestamps are returned in a result buffer. To retrieve information from the result buffer, the following sequence of calls should be made:

NWDSGetAttrValFlags
NWDSGetAttrValModTime
NWDSGetAttrVal

NCP Calls

None

NWDSGetBinderyContext

Returns the setting of the bindery context set on the server identified by connHandle.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetBinderyContext  (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   pnuint8             BinderyEmulationContext);
```

Pascal

```
uses netwin32

Function NWDSGetBinderyContext
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   BinderyEmulationContext : pnuint8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(IN) Specifies the NetWare server connection handle.

binderyEmulationContext

(OUT) Points to a bindery context string.

Return Values

These are common return values.

0x0000 0000	SUCCESSFUL
0x89FE	BAD_PACKET
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The connHandle parameter is the connection handle to the server in which you are interested.

The binderyEmulationContext parameter must have sufficient space allocated to receive the value. For partial dot names, the DCV_CANONICALIZE_NAMES flag determines whether an RDN or a DN is returned for the bindery context. See Section 1.1, “Context Handles,” on page 15 for more information.

The bindery context specifies a location in eDirectory where a bindery connection is allowed to see objects in eDirectory. A bindery connection can see objects only in the eDirectory container defined by the server’s bindery context.

Bindery context is set on NetWare 4.x and 5.x servers by using the `SET BINDERY CONTEXT` command at the server console.

NCP Calls

0x2222 104 04 Return Bindery Context

See Also

NWDSAuditGetObjectID (obsolete 06/03) (page 99)

NWDSGetClassDef

Retrieves an object-class definition from a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetClassDef  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              className,
   pClass_Info_T       classInfo);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSGetClassDef is used to retrieve class definitions from a result buffer filled in by NWDSReadClassDef.

The className parameter 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 Unicode characters (Unicode characters are always 16 bits). 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.

For the complete steps on retrieving class information, see “Reading a Class Definition” on page 70.

NCP Calls

None

See Also

NWDSGetClassDefCount (page 185), NWDSGetClassItem (page 187), NWDSGetClassItemCount (page 189), NWDSReadClassDef (page 333)

NWDSGetClassDefCount

Returns the number of object-class definitions stored in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetClassDefCount  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            classDefCount);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

For the complete steps on retrieving class information from a result buffer, see “Reading a Class Definition” on page 70.

NCP Calls

None

See Also

NWDSGetClassDef (page 183)

NWDSGetClassItem

Returns the name of the next object class item stored in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetClassItem  (
  (NWDSContextHandle   context,
   pBuf_T              buf,
  pnstr8               itemName);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The buf parameter points to a Buf_T filled in by NWDSReadClassDef.

The itemName parameter 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 first two lists contain the names of classes. The remaining lists contain the names of attributes.

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 Unicode characters (Unicode characters are always 16 bits). 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 on reading class-definition information, see “Reading a Class Definition” on page 70.

NCP Calls

None

See Also

NWDSGetClassDef (page 183), NWDSGetClassItemCount (page 189), NWDSListContainableClasses (page 258), NWDSReadClassDef (page 333)

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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetClassItemCount  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            itemCount);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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” on page 70.

NCP Calls

None

See Also

NWDSGetClassDef (page 183), NWDSGetClassItem (page 187), NWDSListContainableClasses (page 258), NWDSReadClassDef (page 333)

NWDSGetContext

Returns information about an NDS context handle.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetContext  (
   NWDSContextHandle   context,
   nint                key,
   nptr                value);
```

Pascal

```
uses netwin32

Function NWDSGetContext
  (context : NWDSContextHandle;
   key : nint;
   value : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the context handle to be queried.

key

(IN) Specifies the information to be retrieved (see Section 5.6, “Context Keys and Flags,” on page 467).

value

(OUT) Points to the context handle information.

Return Values

0x0000 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Applications cannot directly access context handle information. Applications must use the NWDSGetContext function to retrieve context information.

The key parameter specifies the type of information to retrieve, and the value parameter points to the information retrieved. The value parameter must point to a variable that matches the data type specified by the key parameter. For data types and defined keys, see Section 5.6, “Context Keys and Flags,” on page 467.

The NWDSGetContext function must be called repetitively to retrieve information contained in multiple keys.

NCP Calls

None

See Also

NWDSCreateContextHandle (page 133), NWDSSetContext (page 387)

NWDSGetCountByClassAndName

Counts the immediate subordinates of an eDirectory object, restricting the count to objects of a specified object class, with a specific name, or both.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetCountByClassAndName  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              className,
   pnstr8              subordinateName,
   pnint32             count);
```

Pascal

```
uses netwin32

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 base class 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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory 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, the count of immediate subordinate objects is restricted by both filters.
- 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 is restricted only by the className filter.
- When subordinateName is provided and className is NULL, the count of immediate subordinates is restricted only by the subordinateName filter.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSListByClassAndName (page 254)

NWDSGetCurrentUser

Returns the handle of an eDirectory user.

Local Servers: nonblocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: NDS

Syntax

```
#include <nwconn.h>
#include <nwdsapi.h>

 int NWDSGetCurrentUser (
   void);
```

Return Values

Returns the user handle of the current user.

Remarks

This function returns the current eDirectory user in the Thread Group Control Structure (TGCS). The function is valid only on the NLM platform. The current user determines which authentication information is used. For more information, see “Establishing Identities to Multiple eDirectory Trees—NLM Platform” on page 55.

See Also

NWDSSetCurrentUser (page 389)

NWDSGetDefNameContext

Retrieves the default name context for a specified tree.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSGetDefNameContext  (
   NWDSContextHandle   context,
   nuint               nameContextLen,
   pnstr8              nameContext);
```

Pascal

```
uses netwin32

Function NWDSGetDefNameContext
  (context : NWDSContextHandle;
   nameContextLen : nuint;
   nameContext : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request or NULL for the preferred tree.

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 can return the name context in Unicode while NWGetDefaultNameContext could only return the data in local code page format.

The DCV_XLATE_STRINGS flag determines whether local code page format or Unicode strings are returned. For more information, see "DCK_FLAGS Key" on page 18.

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 tree name is not set in the context, the preferred tree will be used. For requesters that do not support multiple trees, if the tree name is specified (a NULL string is not returned to the eDirectory libraries) and if the tree name is different from the preferred tree, an error will be returned.

NCP Calls

None

See Also

NWGetDefaultNameContext (page 410), NWSetDefaultNameContext (page 427), NWDSSetDefNameContext (page 390)

NWDSGetDSIInfo

Returns DSI object information not stored in the attributes of an object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSGetDSIInfo  (
   NWDSContextHandle   context,
   nptr                buf,
   nuint32             bufLen,
   nuint32             infoFlag,
   nptr                data);
```

Pascal

```
uses netwin32

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 or NWDSSearch functions.

bufLen

(IN) Specifies the length of the buf parameter.

infoFlag

(IN) Specifies the data element to be extracted from the buffer pointed to in the buf parameter (see Section 5.11, “DCK_DSI_FLAGS Values,” on page 472).

data

(OUT) Points to a buffer to receive the data element value.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The NWDSReadObjectDSIInfo function returns data regarding an eDirectory object. The NWDSGetDSIInfo function extracts 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 Section 5.11, “DCK_DSI_FLAGS Values,” on page 472. The buffer pointed to by the data parameter must be large enough for the data type of the element.

The DSI information can be retrieved in any order and does not have to follow the order of the DSI_OUTPUT_FIELDS flag.

Object information can be useful to applications browsing the eDirectory tree.

NCP Calls

None

See Also

NWDSGetObjectNameAndInfo (page 217), NWDSReadObjectDSIInfo (page 338)

NWDSGetDSVerInfo

Returns NDS/eDirectory version information.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetDSVerInfo  (
   NWCONN_HANDLE   conn,
   pnuint32        dsVersion,
   pnuint32        rootMostEntryDepth,
   pnstr8          sapName,
   pnuint32        flags,
   punicode        treeName);
```

Pascal

```
uses netwin32

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 eDirectory server.

dsVersion

(OUT) Points to the DS.NLM build version.

rootMostEntryDepth

(OUT) Points to 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) Returns 0x00000001 if the server's root-most partition is a master replica.

treeName

(OUT) Points to the "enabled" tree name (in the Unicode character set).

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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).

To return the letter with the NDS/eDirectory version, use the NWDSReadNDSInfo function.

NCP Calls

None

See Also

NWDSReadNDSInfo (page 336), NWGetNWNetVersion (page 415)

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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsacl.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetEffectiveRights  (
   NWDSContextHandle   context,
   pnstr8              subjectName,
   pnstr8              objectName,
   pnstr8              attrName,
   pnuint32            privileges);
```

Pascal

```
uses netwin32

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 granted.

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 granted to subjectName (see Section 5.18, “eDirectory Access Control Rights,” on page 477).

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory is granted through access control lists (ACLs). The ACL is an attribute defined by the eDirectory 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 eDirectory, or it can be one of the following "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.

The attrName parameter 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:

```
[All Attributes Rights]
[Entry Rights]
```

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 eDirectory NCP

0x2222 104 02 Send eDirectory Fragmented Request/Reply

NWDSGetMonitoredConnRef

Retrieves a monitored connection reference.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSGetMonitoredConnRef  (
   NWDSContextHandle   context,
   pnuint32            connRef);
```

Pascal

```
uses netwin32

Function NWDSGetMonitoredConnRef
  (context : NWDSContextHandle;
   Var connRef : nuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request or NULL for the preferred tree.

connRef

(OUT) Points to the connection reference.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 tree name is not set in the context, the preferred tree will be used. For requesters that do not support multiple trees, if the tree name is specified (a NULL string is not returned to the eDirectory libraries) and if the tree name is different from 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 (page 299), NWCCOpenConnByRef (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk661.html)

NWDSGetNDSInfo

Retrieves NDSPING information.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetNDSInfo  (
   NWDSContextHandle   context,
   pBuf_T              resultBuffer,
   nflag32             requestedField,
   nptr                data);
```

Pascal

```
uses netwin32

Function NWDSGetNDSInfo
  (context : NWDSContextHandle;
   resultBuffer : pBuf_T;
   requestedField : nflag32;
   data : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

resultBuffer

(IN) Points to the buffer filled by a call to the NWDSReadNDSInfo function.

requestedField

(IN) Specifies the DSPING flag for which information is needed (see Section 5.19, “eDirectory Ping Flags,” on page 479).

data

(OUT) Points to information specified in requestedField.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSGetNDSInfo retrieves DSPING information, one field per call, returned in the resultBuffer of NWDSReadNDSInfo. The fields in the resultBuffer can be called in any order, and information in a particular field can be retrieved multiple times. Retrieval can continue until the resultBuffer has been reused or freed.

For instructions on how to use NWDSGetNDSInfo, see “Accessing eDirectory Ping Information” on page 54.

NCP Calls

None

See Also

NWDSReadNDSInfo (page 336),

NWDSGetObjectCount

Returns the number of objects whose information is stored in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetObjectCount  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            objectCount);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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. See these functions for more information.

NCP Calls

None

See Also

NWDSGetAttrName (page 173), NWDSGetAttrVal (page 175), NWDSGetObjectName (page 214), NWDSList (page 248), NWDSRead (page 327), NWDSSearch (page 383)

NWDSGetObjectHostServerAddress

Returns the addresses of the server where an object is located.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetObjectHostServerAddress  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              serverName,
   pBuf_T              netAddresses);
```

Pascal

```
uses netwin32

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 eDirectory 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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSGetObjectHostServerAddress works only for objects having "Host Server" as an attribute (such as Volume objects). Servers can have more than one address, such as an IPX and an IP address. The netAddresses parameter receives these addresses.

For information on retrieving the addresses from the buffer, see “Finding the Host Server of an Object” on page 60.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSComputeAttrValSize (page 130), NWDSGetAttrCount (page 169), NWDSGetAttrVal (page 175)

NWDSGetObjectName

Returns the name and information about the next object whose information is stored in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetObjectName  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              objectName,
   pnuint32            attrCount,
   pObject_Info_T      objectInfo);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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.

Use the NWDSGetObjectName function when the context handle has the default values for the DCK_DSI_FLAGS. If you have changed the requested DSI information, use the NWDSGetObjectNameAndInfo function to retrieve the information. For more information, see "DCK_DSI_FLAGS Key" on page 21.

NOTE: You must retrieve all of the information about the current object before calling NWDSGetObjectName for the next object.

The buf parameter points to a Buf_T filled in by NWDSList, NWDSRead, or NWDSSearch.

The objectName parameter 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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

The attrCount parameter 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 returns only the names of objects. The attribute count will be zero or greater for a buffer returned by NWDSSearch.

The objectInfo parameter points to additional information about the object. You must allocate memory to retrieve this information (sizeof(Object_Info_T)).

For more information, see NWDSList, NWDSRead, or NWDSSearch.

NCP Calls

None

See Also

NWDSGetAttrName (page 173), NWDSGetAttrVal (page 175), NWDSGetObjectCount (page 210), NWDSList (page 248), NWDSRead (page 327), NWDSSearch (page 383)

NWDSGetObjectNameAndInfo

Returns the name and information about the next object whose information is stored in a result buffer.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSGetObjectNameAndInfo  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              objectName,
   pnuint32            attrCount,
   ppnstr8             objectInfo);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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.

Use the NWDSGetObjectNameAndInfo function when you have defined the return values for the DCK_DSI_FLAGS in the context handle. If you are using default values, use the NWDSGetObjectName function to retrieve the information. For more information, see "DCK_DSI_FLAGS Key" on page 21

NOTE: You must retrieve all of the information about the current object before calling NWDSGetObjectNameAndInfo for the next object.

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 Unicode characters (Unicode characters are always 16 bits). 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 returns only 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. Use the NWDSGetDSIInfo function to read the information in the buffer pointed to by objectInfo. Use this pointer as the buf parameter for the NWDSGetDSIInfo function.

For more information, see the NWDSList, NWDSRead, and NWDSSearch functions.

NCP Calls

None

See Also

NWDSGetAttrName (page 173), NWDSGetAttrVal (page 175), NWDSGetDSIInfo (page 199), NWDSGetObjectCount (page 210), NWDSReadObjectDSIInfo (page 338), NWDSList (page 248), NWDSRead (page 327), NWDSSearch (page 383)

NWDSGetPartitionExtInfo

Retrieves replica information from a result buffer filled by the NWDSListPartitionsExtInfo function.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#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

```
uses netwin32

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 NWDSListPartitionsExtInfo 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 buffer pointed to by the infoPtr parameter. For a list of possible flags, see Section 5.20, “DSP Replica Information Flags,” on page 481.

length

(IN) Points to the size of the returned information.

data

(OUT) Points to the returned information.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Call NWDSGetPartitionExtInfo repeatedly to access additional information stored in the infoPtr parameter.

For the complete steps for retrieving partition information see “Listing Partitions and Retrieving Partition Information” on page 66.

NCP Calls

None

See Also

NWDSGetPartitionExtInfoPtr (page 222), NWDSGetServerName (page 234), NWDSListPartitions (page 264), NWDSListPartitionsExtInfo (page 267)

NWDSGetPartitionExtInfoPtr

Retrieves a pointer to the replica information from a result buffer filled by the NWDSListPartitionsExtInfo function.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSGetPartitionExtInfoPtr  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   ppnstr8             infoPtr,
   ppnstr8             infoPtrEnd);
```

Pascal

```
uses netwin32

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 NWDSListPartitionsExtInfo function.

infoPtr

(OUT) Points to the returned information.

infoPtrEnd

(OUT) Points to the end of the returned information.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

See Section 5.20, “DSP Replica Information Flags,” on page 481.

For the complete steps for retrieving partition information, see the “Listing Partitions and Retrieving Partition Information” on page 66.

NCP Calls

None

See Also

NWDSGetPartitionExtInfo (page 220), NWDSGetServerName (page 234), NWDSListPartitions (page 264), NWDSListPartitionsExtInfo (page 267)

NWDSGetPartitionInfo

Retrieves replica information from a result buffer filled by NWDSListPartitions.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetPartitionInfo  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              partitionName,
   pnuint32            replicaType);
```

Pascal

```
uses netwin32

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 (see Section 5.23, “Replica Types,” on page 483).

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The buf parameter points to a Buf_T filled by NWDSListPartitions.

The partitionName parameter 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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

The replicaType parameter points to NWREPLICA_TYPE containing information about the type of replica the partition is. The replica types are enumerated in NWDSDEFS.H (for a description, see Section 5.23, “Replica Types,” on page 483).

For the complete steps for retrieving partition information, see “Listing Partitions and Retrieving Partition Information” on page 66.

NCP Calls

None

See Also

NWDSGetServerName (page 234), NWDSListPartitions (page 264)

NWDSGetPartitionRoot

Returns the partition root name of the given object.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetPartitionRoot  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              partitionRoot);
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

NWDSGetServerAddresses (obsolete 3/98)

Returns the network addresses of the server associated with a connection handle. This function is now obsolete. Call NWDSGetServerAddresses2 (page 230) instead.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetServerAddresses  (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   pnuint32            countNetAddress,
   pBuf_T              netAddresses);
```

Pascal

```
uses netwin32

Function NWDSGetServerAddresses
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   countNetAddress : pnuint32;
   netAddresses : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the eDirectory 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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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" on page 57.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply
53 Get Server Address

See Also

NWDSComputeAttrValSize (page 130), NWDSGetAttrCount (page 169), NWDSGetAttrVal (page 175)

NWDSGetServerAddresses2

Returns the network addresses of the server associated with a connection handle.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetServerAddresses2  (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   pnuint32            countNetAddress,
   pBuf_T              netAddresses);
```

Pascal

```
uses netwin32

Function NWDSGetServerAddresses2
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   countNetAddress : pnuint32;
   netAddresses : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(IN) Specifies the connection handle for the target server.

countNetAddress

(OUT) Points to the number of network addresses contained in the netAddresses parameter.

netAddresses

(OUT) Points to a buffer containing the network address associated with the server.

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Servers can have more than one address, such as an IPX and an IP address. The netAddresses parameter receives these addresses.

For more information, see “Retrieving Addresses of a Connected Server” on page 57.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply
53 Get Server Address

See Also

NWDSComputeAttrValSize (page 130), NWDSGetAttrVal (page 175)

NWDSGetServerDN

Returns the server's distinguished name.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetServerDN  (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   pnstr8              serverDN);
```

Pascal

```
uses netwin32

Function NWDSGetServerDN
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   serverDN : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(IN) Specifies the connection to the server to be queried.

serverDN

(OUT) Points to the distinguished name of the server.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The connHandle parameter 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 Unicode characters (Unicode characters are always 16 bits). 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. For more information, see Section 1.1, “Context Handles,” on page 15.

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 ERR_NO_CONNECTION error is returned.

NCP Calls

0x2222 104 02 Send eDirectory 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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetServerName  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              serverName,
   pnuint32            partitionCount);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSGetServerName should be the first function called to read from a result buffer returned by NWDSListPartitions.

The buf parameter points to a Buf_T filled by NWDSListPartitions.

The serverName parameter 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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

For the complete steps for retrieving partition information, see “Listing Partitions and Retrieving Partition Information” on page 66.

NCP Calls

None

See Also

NWDSGetPartitionInfo (page 224), NWDSListPartitions (page 264)

NWDSGetSyntaxCount

Returns the number of eDirectory syntaxes whose information is stored in a result buffer filled by NWDSReadSyntaxes.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetSyntaxCount  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnuint32            syntaxCount);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

For complete steps on retrieving information about the syntaxes in the eDirectory schema, see “Retrieving Syntax Names and Definitions” on page 73.

NCP Calls

None

See Also

NWDSGetSyntaxDef (page 238), NWDSReadSyntaxes (page 348)

NWDSGetSyntaxDef

Retrieves the next eDirectory-syntax definition from a result buffer filled by NWDSReadSyntaxes.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetSyntaxDef  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              syntaxName,
   pSyntax_Info_T      syntaxDef);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

The buf parameter points to a result buffer containing information about syntaxes. This result buffer is allocated by NWDSAllocBuf and filled by NWDSReadSyntaxes.

The syntaxName parameter points to the name of the syntax 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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

The syntaxDef parameter 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 must be NULL.

The user must allocate memory, sizeof(Syntax_Info_T), to receive the syntax definition.

For the complete steps on retrieving information about the syntaxes in the eDirectory schema, see “Retrieving Syntax Names and Definitions” on page 73.

NCP Calls

None

See Also

NWDSGetSyntaxCount (page 236), NWDSReadSyntaxes (page 348)

NWDSGetSyntaxID

Returns the syntax ID of a given attribute.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSGetSyntaxID (
   NWDSContextHandle   context,
   pnstr8              attrName,
   pnuint32            syntaxID);
```

Pascal

```
uses netwin32

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 (see Section 5.26, “Syntax IDs,” on page 487).

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Syntax IDs are enumerated in the nwdsdefs.h file. A description of syntax definitions can be found in “Attribute Syntax Definitions” (*NDK: Novell eDirectory Schema Reference*).

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

NWDSInitBuf

Initializes a buffer for use as a request buffer for an eDirectory function.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSInitBuf  (
   NWDSContextHandle   context,
   nuint32             operation,
   pBuf_T              buf);
```

Pascal

```
uses netwin32

Function NWDSInitBuf
  (context : NWDSContextHandle;
   operation : nuint32;
   buf : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

operation

(IN) Specifies the eDirectory operation for which the buffer is being initialized (see Section 5.3, “Buffer Operation Types and Related Functions,” on page 464).

buf

(IN) Points to the buffer being initialized.

Return Values

0x0000 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

The buffer pointed to by the buf parameter is updated to reflect the selected operation.

NCP Calls

None

See Also

NWDSAllocBuf (page 95), NWDSFreeBuf (page 158)

NWDSInspectEntry

Inspects an object for correctness.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSInspectEntry  (
   NWDSContextHandle   context,
   pnstr8              serverName,
   pnstr8              objectName,
   pBuf_T              errBuffer);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSInspectEntry is a diagnostic function allowing you to inspect an object in the replica 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.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

NWDSJoinPartitions

Joins a subordinate partition to its parent partition.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSJoinPartitions  (
   NWDSContextHandle   context,
   pnstr8              subordinatePartition,
   nflag32             flags);
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAddReplica (page 91), NWDSChangeReplicaType (page 122), NWDSSplitPartition (page 394)

NWDSList

Lists the immediate subordinates of an object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSList  (
   NWDSContextHandle   context,
   pnstr8              object,
   pnint_ptr           iterationHandle,
   pBuf_T              subordinates);
```

Pascal

```
uses netwin32

Function NWDSList
  (context : NWDSContextHandle;
   object : pnstr8;
   iterationHandle : pnint_ptr;
   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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSList returns an object's base class, entry flags, modification time, and subordinate object count.

The function succeeds if the object specified by object is found in eDirectory, regardless of whether there is any subordinate information to return.

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. For more information, see Section 5.6, “Context Keys and Flags,” on page 467.

The results buffer pointed to by subordinates receives a sequence of Object_Info_T (page 452) structures containing information about objects subordinate to the specified object.

The iterationHandle parameter controls the retrieval of list results larger than the result buffer pointed to by subordinates.

Before the initial call to NWDSList, set the iterationHandle parameter to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when NWDSList 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 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.

NOTE: On large networks, iterative processes, such as NWDSList, might take a long time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. Developers should use NWDSCloseIteration to allow users of their applications to abort an iterative process that is taking too long to complete.

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 eDirectory Container" on page 60.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSSearch (page 383)

NWDSListAttrsEffectiveRights

Returns an object's effective privileges on another object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsacl.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSListAttrsEffectiveRights  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              subjectName,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint_ptr           iterationHandle,
   pBuf_T              privilegeInfo);
```

Pascal

```
uses netwin32

Function NWDSListAttrsEffectiveRights
  (context : NWDSContextHandle;
   objectName : pnstr8;
   subjectName : pnstr8;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint_ptr;
   privilegeInfo : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the eDirectory object whose access rights are to be checked.

subjectName

(IN) Points to the name of the eDirectory 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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The subjectName parameter is the name of an eDirectory object. If subjectName is NULL, the name of the currently logged-in object is used.

The allAttrs and attrNames parameters 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.

The attrNames parameter points to a request buffer explicitly specifying the names of the attributes for which information is to be returned.

The iterationHandle parameter 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.

NOTE: On large networks, iterative processes, such as NWDSListAttrsEffectiveRights, might take a long time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. Developers should use NWDSCloseIteration to allow users of their applications to abort an iterative process that is taking too long to complete.

To end the list operation before the complete results have been retrieved, call NWDSCloseIteration with a value of DSV_READ to free memory and states associated with the List operation.

For more information, see "Determining the Effective Rights of an Object" on page 59.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSAllocBuf (page 95), NWDSGetAttrVal (page 175), NWDSInitBuf (page 242)

NWDSListByClassAndName

Lists the immediate subordinates for an eDirectory 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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSListByClassAndName  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              className,
   pnstr8              subordinateName,
   pnint_ptr           iterationHandle,
   pBuf_T              subordinates ;)
```

Pascal

```
uses netwin32

Function NWDSListByClassAndName
  (context : NWDSContextHandle;
   objectName : pnstr8;
   className : pnstr8;
   subordinateName : pnstr8;
   iterationHandle : pnint_ptr;
   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.

0x0000 0000	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 eDirectory
0xFE0D	UNI_NO_DEFAULT
0xFE0F	UNI_HANDLE_MISMATCH
0xFEB5	ERR_NULL_POINTER
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSListByClassAndName controls the list output with filters on the class, on the name, or on both.

If the context handle is set for partial dot names and the 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. For more information, see Section 1.1, “Context Handles,” on page 15.

The name given for the className ’s filter is the name of an object's base class, such as User, Computer, or NCP Server.

The value given for the 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 eDirectory tree is immediately subordinate to the object specified by objectName. It is not relative to the current name context in eDirectory specified by context.

The relationship between className and subordinateName is an "AND" relationship.

- When className and subordinateName are provided, the list of immediate subordinate objects is restricted by both filters.
- When className is NULL and subordinateName is NULL, all the immediate subordinates are returned.
- When className is provided and subordinateName is NULL, the list of immediate subordinates restricted only by className’s filter.
- When className is NULL and subordinateName is provided, the list of immediate subordinates is restricted only by subordinateName’s filter.

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."
```

The iterationHandle parameter controls retrieval of search results larger than the result buffer pointed to by subordinates.

Before the initial call to NWDSListByClassAndName, set the iterationHandle parameter 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.

NOTE: On large networks, iterative processes, such as 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. Developers should use NWDSCloseIteration to allow users of their applications to abort an iterative process that is taking too long to complete.

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.

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 on reading the results, see "Retrieving Results from eDirectory Output Buffers" on page 53.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSList (page 248)

NWDSListContainableClasses

Returns the names of the object classes that can be contained by (subordinate to) the specified object in the eDirectory tree.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdssch.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSListContainableClasses  (
   NWDSContextHandle   context,
   pnstr8              parentObject,
   pnint_ptr           iterationHandle,
   pBuf_T              containableClasses);
```

Pascal

```
uses netwin32

Function NWDSListContainableClasses
  (context : NWDSContextHandle;
   parentObject : pnstr8;
   iterationHandle : pnint_ptr;
   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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

The parentObject parameter points to the name of an eDirectory object for which containable object-classes are to be listed. If this parent object is not a valid container object, an error is returned.

The iterationHandle parameter 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.

If the result buffer holds the complete results when NWDSListContainableClasses 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 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.

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.

The containableClasses parameter 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 eDirectory Schema (those from which objects can be created). Alias is always included in the list.

For more information, see “Listing Containable Classes” on page 69.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSAddObject (page 87)

NWDSListContainers

Lists container objects subordinate to a specific eDirectory object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSListContainers  (
   NWDSContextHandle   context,
   pnstr8              object,
   pnint_ptr           iterationHandle,
   pBuf_T              subordinates);
```

Pascal

```
uses netwin32

Function NWDSListContainers
  (context : NWDSContextHandle;
   object : pnstr8;
   iterationHandle : pnint_ptr;
   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.

0x0000 0000	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 eDirectory
0xFE01	ERR_BAD_CONTEXT
0xFE0D	UNI_NO_DEFAULT
0xFE0F	UNI_HANDLE_MISMATCH
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The name specified by objectName is relative to the current name context in context if partial dot format is used and the DCV_CANONICALIZE_NAMES flag is set. It can be typed or untyped. For more information, see Section 1.1, “Context Handles,” on page 15.

iterationHandle controls retrieval of search results larger than the result buffer pointed to by subordinates.

Before the initial call to NWDSListContainers, set the iterationHandle parameter 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.

NOTE: On large networks, iterative processes, such as NWDSListContainers, might take a long time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. Developers should use NWDSCloseIteration (page 126) to allow users of their applications to abort an iterative process that is taking too long to complete.

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 on reading the results, see “Retrieving Results from eDirectory Output Buffers” on page 53.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSList (page 248)

NWDSListPartitions

Returns information about the replicas of partitions stored on the specified server.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSListPartitions  (
   NWDSContextHandle   context,
   pnint_ptr           iterationHandle,
   pnstr8              server,
   pBuf_T              partitions);
```

Pascal

```
uses netwin32

Function NWDSListPartitions  (
   context : NWDSContextHandle;
   iterationHandle : pnint_ptr;
   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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The iterationHandle parameter controls retrieval of results larger than the result buffer pointed to by subordinates.

Before the initial call to NWDSListPartitions, set the contents of the iteration handle pointed to by iterationHandle to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when NWDSListPartitions returns from its initial call,, 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 NWDSListPartitions 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_LIST_PARTITIONS to free memory and states associated with the list operation.

For more information, see “Listing Partitions and Retrieving Partition Information” on page 66.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSGetServerName (page 234), NWDSGetPartitionInfo (page 224)

NWDSListPartitionsExtInfo

Returns extended information about the replicas stored on the specified server.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_GLOBAL_LIBRARY (NWDSCCODE) NWDSListPartitionsExtInfo  (
   NWDSContextHandle   context,
   pnint_ptr           iterationHandle,
   pnstr8              server,
   nflag32             DSPFlags,
   pBuf_T              partitions);
```

Pascal

```
uses netwin32

Function NWDSListPartitionsExtInfo
  (context : NWDSContextHandle;
   iterationHandle : pnint_ptr;
   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 information flags (see Section 5.20, "DSP Replica Information Flags," on page 481).

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

The iterationHandle parameter controls retrieval of results larger than the result buffer pointed to by subordinates.

Before the initial call to NWDSListPartitionsExtInfo, set the contents of the iteration handle pointed to by iterationHandle to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when NWDSListPartitionsExtInfo returns from its initial call, 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 NWDSListPartitionsExtInfo 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_LIST_PARTITIONS to free memory and states associated with the list operation.

For more information, see "Listing Partitions and Retrieving Partition Information" on page 66.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSGetPartitionExtInfo (page 220), NWDSGetPartitionExtInfoPtr (page 222), NWDSGetServerName (page 234), NWDSListPartitions (page 264)

NWDSLogin

Performs all authentication operations needed to establish a client's connection to the network and to the network's authentication service. Does not support international or extended characters in passwords.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSLogin  (
   NWDSContextHandle   context,
   nflag32             optionsFlag,
   pnstr8              objectName,
   pnstr8              password,
   nuint32             validityPeriod);
```

Pascal

```
uses netwin32

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 which authentication will be valid with other servers. Pass in zero (0).

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values".

Remarks

NWDSLogin caches authentication information locally to be used by other functions and in background authentication to additional services.

The password parameter points to the client's current password in clear text. If there is no password for the client, its value 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.

Until an authenticated connection is established, the client can access only eDirectory information classified as public.

NOTE: NWDSLoginEx (page 272) supports internation and extended characters in passwords and is recommended in place of NWDSLogin.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAuthenticate (obsolete 06/03) (page 101), NWDSLogout (page 275), NWDSLoginEx (page 272)

NWDSLoginEx

Performs all authentication operations needed to establish a client's connection to the network and to the network's authentication service. Supports international and extended characters in passwords.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSLoginEx  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   nuint32             pwdFormat,
   nptr                pwd);
```

Pascal

```
uses netwin32

Function NWDSLoginEx
  (context : NWDSContextHandle;
   objectName : pnstr8;
   pwdFormat : nuint32;
   pwd : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the name of the object logging into the network.

pwdFormat

(IN) Specifies the format of the password data. Select from the following:

PWD_UNICODE_STRING
PWD_UTF8_STRING
PWD_RAW_C_STRING

pwd

(IN) Points to the client's password.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values".

Remarks

NWDSLogin caches authentication information locally to be used by other functions and in background authentication to additional services.

The password parameter points to the client's current password in clear text. If there is no password for the client, its value 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.

Until an authenticated connection is established, the client can access only eDirectory information classified as public.

NOTE: The PWD_RAW_C_STRING password format allows any arbitrary NULL-terminiated data to be used as a password. Passwords specified with this format are not interoperable with unicode and UTF8 passwords.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAuthenticateConn (page 103), NWDSAuthenticateConnEx (page 105), NWDSLogout (page 275)

NWDSLoginAsServer

Allows an NLM application to log in to eDirectory as if it were the NCP Server object.

Local Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM

Service: NDS

Syntax

```
#include <nwdsapi.h>

N_GLOBAL_LIBRARY(NWCCODE)  NWDSLoginAsServer (
   NWDSContextHandle    context);
```

Parameters

context

(IN) Specifies the NDS context for the request.

Return Values

0x0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (-001 to -799).

Remarks

This function is available for the NLM platform only. It logs in the NLM application to eDirectory as the NCP Server object that the NLM is currently running on.

This function cannot be used on remote servers.

NCP Calls

None

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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSLogout  (
   NWDSContextHandle   context);
```

Pascal

```
uses netwin32

Function NWDSLogout
  (context : NWDSContextHandle
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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, although all rights associated with server attachments and session connections are lost.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAuthenticate (obsolete 06/03) (page 101), NWDSLogin (page 270)

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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSMapIDToName  (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   nuint32             objectID,
   pnstr8              object);
```

Pascal

```
uses netwin32

Function NWDSMapIDToName
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   objectID : nuint32;
   objectName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

The connHandle parameter contains a server connection handle. This identifies the server from which the object ID was obtained.

The objectID parameter contains the object ID returned by the specified server.

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 eDirectory object. (If an application needs a short-hand representation of an object, it should manage its own local name-to-ID mapping.)

The object parameter receives the name of the eDirectory 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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSMapNameToID (page 279)

NWDSMapNameToID

Returns the object ID for an eDirectory object on a specified server.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSMapNameToID  (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   pnstr8              object,
   pnuint32            objectID);
```

Pascal

```
uses netwin32

Function NWDSMapNameToID
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   objectName : pnstr8;
   objectID : pnuint32
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

connHandle

(IN) Specifies the connection handle for the target server.

object

(IN) Points to the eDirectory object name.

objectID

(OUT) Points to the object ID for the specified object.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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 eDirectory object. (If an application needs a short-hand representation of an object, it should manage its own local name-to-ID mapping.)

The context parameter specifies a location within the eDirectory tree. The NWDSMapNameToID function is designed for local name conversion when the context parameter uses default values (see Section 5.7, "Default Context Key Values," on page 469). If the context has been set to nondefault values (such as not dereferencing aliases) with the NWDSSetContext (page 387) function, the NWDSMapNameToID function should not be used. Instead, use the NWDSReadObjectDSIInfo (page 338) and NWDSGetDSIInfo (page 199) functions.

The connHandle parameter contains a server connection handle. It identifies the server from which the object ID is to be obtained.

The object parameter points to the name of the eDirectory 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.

The objectID parameter points to the object ID of the specified name on the server identified by conn.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSMapIDToName (page 277)

NWDSModifyClassDef

Modifies an existing object class definition.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdssch.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSModifyClassDef  (
   NWDSContextHandle   context,
   pnstr8              className,
   pBuf_T              optionalAttrs);
```

Pascal

```
uses netwin32

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 optional attribute list for the object class.

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

The className parameter identifies the object class to which optional attributes will be added.

The optionalAttrs parameter points to a request buffer containing a list of attribute names to be added to the optional attribute 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSDefineClass (page 137)

NWDSModifyDN

Changes the distinguished name of an object or its alias in the eDirectory tree.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSModifyDN  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              newDN,
   nbool8              deleteOldRDN);
```

Pascal

```
uses netwin32

Function NWDSModifyDN
  (context : NWDSContextHandle;
   objectName : pnstr8;
   newDN : pnstr8;
   deleteOldRDN : nbool8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request and the name format for the objectName parameter.

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 RDN. If FALSE, the old RDN is retained as an additional attribute value. If TRUE, the old RDN is deleted.

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The DN is the name of the object that includes the path from the object to the root container in the eDirectory tree. The object being modified must be a leaf object, but it may be either an object or its alias.

The objectName parameter points to the object whose DN is to be modified. Aliases in the name will not be dereferenced.

The newDN parameter specifies the new DN of the object. For example:

```
"CN=Mary.OU=Graphics.O=WimpleMakers"
```

If the container objects in the DN are different from the object's current path, the object is moved to the new container.

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 old attribute values in the RDN are deleted. If FALSE, old values remain in the object (but not as a part of the DN). If the naming attribute is single valued, this flag must be TRUE.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSModifyObject (page 286), NWDSSetContext (page 387), NWDSGetContext (page 191)

NWDSModifyObject

Modifies an object or its alias.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSModifyObject  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnint_ptr           iterationHandle,
   nbool8              more,
   pBuf_T              changes);
```

Pascal

```
uses netwin32

Function NWDSModifyObject
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint_ptr;
   more : nbool8;
   changes : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(IN) Points to the object to be modified.

iterationHandle

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

more

(IN) Specifies whether additional information will be returned:

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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

The objectName parameter 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, or NWDSPutChangeandVal.

NOTE: If the iterationHandle parameter is set to 0 initially, NWDSModifyObject will ignore the value and process the request as if -1 was passed.

You can set up multiple buffers to hold changes to an object and have them processed as one modification. To do this, set the more parameter to a nonzero value. This informs eDirectory that you have multiple changes for the same object. When you send the last buffer of information for the object with NWDSModifyObject, set the more parameter to zero. This signals eDirectory to begin processing all the changes for the object.

If the NDS/eDirectory version does not support this iterative process, ERR_BUFFER_FULL will be returned and you will need to send each change as a separate modification with the more parameter set to zero.

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 eDirectory 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.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSModifyDN (page 283), NWDSRemoveObject (page 359), NWDSModifyRDN (page 289), NWDSMutateObject (page 295)

NWDSModifyRDN

Changes the naming attribute of an eDirectory object or its alias in the eDirectory tree.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSModifyRDN  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              newDN,
   nbool8              deleteOldRDN);
```

Pascal

```
uses netwin32

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. If TRUE, the old RDN is deleted.

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSModifyRDN does not move an object to a new location in the eDirectory tree.

NWDSModifyRDN changes only 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 eDirectory 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 eDirectory tree. (To move an object, call NWDSMoveObject or NWDSModifyDN.)

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 named Hector.

The objectName parameter identifies the object whose name is to be modified. Aliases in the name are not dereferenced.

The newDN parameter specifies the new name of the object. If an attribute value in the new RDN does not already exist in the object, it is added. If it cannot be added, an error is returned.

If deleteOldRDN is TRUE, all old attribute values in the RDN are deleted. If FALSE, old values remain in the object (but not as a part of the DN). If the naming attribute is single valued, this flag must be TRUE.

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 eDirectory NCP

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSModifyDN (page 283), NWDSModifyObject (page 286), NWDSGetContext (page 191), NWDSSetContext (page 387)

NWDSMoveObject

Moves an eDirectory object from one container to another and/or renames the object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSMoveObject  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              destParentDN,
   pnstr8              destRDN);
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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.

The objectName parameter 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.

The context parameter determines the name form for the objectName parameter. For partial dot form names, the DCV_CANONICALIZE_NAMES flag determines whether an RDN or a DN should be entered. For more information, see Section 1.1, "Context Handles," on page 15.

The destParentDN parameter identifies the name of the parent object the moved object is to be directly subordinate to. The parent object must already exist in the eDirectory tree.

The destRDN parameter specifies the new RDN of the object being moved.

If Hector is represented in the eDirectory tree as

```
CN=Hector.OU=Graphics.O=WimpleMakers
```

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 eDirectory tree, and his complete NDS name becomes

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

NWDSMoveObject is similar in functionality to NWDSModifyDN. Both functions can move an object to a new container. Both allow you to change the value of the object's naming attribute. NWDSModifyDN allows you to select whether the old naming attribute value is kept or deleted for multivalued naming attributes. NWDSMoveObject always deletes an old value.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSModifyDN (page 283), NWDSSetContext (page 387), NWDSGetContext (page 191)

NWDSMutateObject

Mutates the specified entry from its current object class to the specified class.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSMutateObject  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnstr8              newObjectClass,
   nuint32             flags);
```

Parameters

context

(IN) Specifies the NDS context for the request or NULL for the preferred tree.

objectName

(IN) Points to the name of the object whose class is being changed.

newObjectClass

(IN) Points to the object class name.

flags

(IN) Specifies whether the default ACL template of the new object class should be applied to the object. The DSM_APPLY_ACL_TEMPLATES flag indicates that the default ACL template of the new class should be applied to the object.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

To mutate an object from one class definition to another, the two definitions need to have matching mandatory attributes and schema rules. For example, a User object can be mutated to an

Organizational Person object because both definitions are effective classes and share the same mandatory attributes, naming attributes, and containment rules. However, the User class definition has many more optional attributes defined than the Organizational Person class. If the object you are mutating has values in some of these attributes, the information will be lost.

If an auxiliary class has been added to the object, the auxiliary class attributes and their values will mutate to the new object.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSModifyObject (page 286)

NWDSOpenConnToNDSServer

Locates a connection to a specific server.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSOpenConnToNDSServer  (
   NWDSContextHandle   context,
   pnstr8              serverName,
   pNWCONN_HANDLE      connHandle);
```

Pascal

```
uses netwin32

Function NWDSOpenConnToNDSServer
 (context : NWDSContextHandle;
   serverName : pnstr8;
   Var connHandle : NWCONN_HANDLE
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request or NULL for the preferred tree.

serverName

(IN) Points to the server to receive the request.

connHandle

(OUT) Points to the connection handle.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSOpenConnToNDSServer allows you to locate a connection to a specific server. The form of the name depends upon the settings in the context handle. If the default values have not been modified, this is a partial dot name relative to the current NDS context. For more information, see Section 1.1, “Context Handles,” on page 15.

The serverName is resolved using eDirectory and a connection is returned.When finished with the connection, the application must close the connection using NWCCCloseConn (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk125.html).

NCP Calls

None

See Also

NWDSAuthenticateConn (page 103), NWDSOpenMonitoredConn (page 299)

NWDSOpenMonitoredConn

Opens a connection handle to a monitored connection.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSOpenMonitoredConn  (
   NWDSContextHandle   context,
   pNWCONN_HANDLE      connHandle);
```

Pascal

```
uses netwin32

Function NWDSOpenMonitoredConn
  (context : NWDSContextHandle;
   Var connHandle : NWCONN_HANDLE
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request or NULL for the preferred tree.

connHandle

(OUT) Points to the connection handle.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

A monitored connection is a connection between a library and eDirectory that is created when the NWDSLogin function is called. This function must establish a connection to a server that has a read/

write replica of the portion on the eDirectory tree where the object logging in resides. eDirectory uses the monitored connection to track concurrent logins to the same tree. An object can be restricted to a specified number of concurrent logins, and the monitored connection allows eDirectory to track how many are allowed and how many are currently in use.

Most of the newer NetWare client requesters control the monitored connection handle, and applications have no need to retrieve information about it or manipulate it.

For multiple tree support, the tree name specified in the context handle is used to specify which monitored connection to retrieve. Call the sForTrees function to return a list of all trees to which you are currently logged in. Call the NWDSSetContext function to set the context for one of the trees returned by the NWDSScanConnsForTrees function. Call NWDSOpenMonitoredConn to open a monitored connection for the same tree. You are responsible for closing the connection handle. If the tree name is not set in the context, the preferred tree will be used.

For requesters that do not support multiple trees, if the tree name is specified (a NULL string is not returned to the eDirectory libraries) and if the tree name is different from the preferred tree, an error will be returned.

NCP Calls

None

See Also

NWDSGetMonitoredConnRef (page 206), NWDSOpenConnToNDSServer (page 297)

NWDSOpenStream

Begins access to an attribute of type SYN_STREAM.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSOpenStream  (
   NWDSContextHandle       context,
   pnstr8                  objectName,
   pnstr8                  attrName,
   nflag32                 flags,
   NWFILE_HANDLE N_FAR    *fileHandle);
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

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.

If the attribute has never been initialized, NWDSOpenStream will return ERR_NO_SUCH_VALUE. The attribute can be initialized when the object is created (NWDSAddObject) or afterwards with the NWDSModifyObject function. Set the attribute's value to NULL.

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 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 NLMs, 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

NWDSPartitionReceiveAllUpdates

Changes the state of the partition so all servers holding a replica will send entire partition information to the specified partition.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPartitionReceiveAllUpdates  (
   NWDSContextHandle   context,
   pnstr8              partitionRoot,
   pnstr8              serverName);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

This function replaces a replica on the specified server when that replica's time stamps are incorrect. For example, an operation to repair a local database may not finish correctly and leave the replica in an unknown state. Use this function to remedy such a problem.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSPartitionSendAllUpdates (page 306)

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.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPartitionSendAllUpdates  (
   NWDSContextHandle   context,
   pnstr8              partitionRoot,
   pnstr8              serverName);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Error conditions may prevent data from being propagated to all replicas of a partition. To remedy this situation, call the NWDSPartitionSendAllUpdates function.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSPartitionReceiveAllUpdates (page 304)

NWDSPutAttrName

Stores an attribute name in a request buffer to be used by an eDirectory function.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPutAttrName  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              attrName);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

This function is used for object operations such as reading, searching, comparing, and adding. It is also used for schema operations such as reading class definitions and attribute definitions. For more information on using this function, see the appropriate task in the “Tasks” on page 49 chapter.

NCP Calls

None

See Also

NWDSAddObject (page 87), NWDSCompare (page 128), NWDSPutAttrVal (page 312), NWDSRead (page 327), NWDSReadAttrDef (page 330), NWDSSearch (page 383)

NWDSPutAttrNameAndVal

Stores an attribute name and value in a request buffer to be used by an eDirectory function.

Local Servers: nonblocking

Remote Servers: nonblocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPutAttrNameAndVal  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              attrName,
   nuint32             syntaxID,
   nptr                attrVal);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

NWDSPutAttrNameAndVal combines the functionality of NWDSPutAttrName and NWDSPutAttrVal and adds error checking so the buffer is always valid. The NWDSPutAttrName and NWDSPutAttrVal functions are used by object operations such as adding an object and comparing attribute values. For these operations, NWDSPutAttrNameAndVal is the preferred function.

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 to add the eDirectory 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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

The buf parameter points to a Buf_T, which is allocated by NWDSAllocBuf and initialized by NWDSInitBuf.

The syntaxID parameter 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 "Attribute Syntax Definitions" (*NDK: Novell eDirectory Schema Reference*).

The attrVal parameter 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 "Attribute Syntax Definitions" to determine the data type associated with an attribute.

NCP Calls

None

See Also

NWDSAddObject (page 87), NWDSCompare (page 128), NWDSModifyObject (page 286), NWDSPutAttrName (page 308), NWDSPutAttrVal (page 312), NWDSRead (page 327), NWDSReadAttrDef (page 330), NWDSSearch (page 383)

NWDSPutAttrVal

Stores an attribute value in a request buffer to be used by an eDirectory function.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPutAttrVal  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   nuint32             syntaxID,
   nptr                attrVal);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The buf parameter 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).

The syntaxID parameter 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 “Attribute Syntax Definitions” (*NDK: Novell eDirectory Schema Reference*).

The attrVal parameter 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 “Attribute Syntax Definitions” to determine the data type associated with an attribute.

NCP Calls

None

See Also

NWDSAddObject (page 87), NWDSAddReplica (page 91), NWDSCompare (page 128), NWDSModifyObject (page 286), NWDSPutAttrName (page 308), NWDSPutChange (page 314), NWDSSplitPartition (page 394)

NWDSPutChange

Stores a change record in a request buffer to be used by NWDSModifyObject.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPutChange  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   nuint32             changeType,
   pnstr8              attrName);
```

Pascal

```
uses netwin32

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 (see Section 5.5, “Change Types for Modifying Objects,” on page 466).

attrName

(IN) Points to the attribute name to be changed.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

A change record includes the name of the attribute and the type of change to be performed.

If an attempt is made to modify the Object Class attribute, an error is returned.

If the change operation requires an attribute value, this function is followed by the NWDSPutAttrVal function which supplies the value for the attribute specified in the NWDSPutChange function. The NWDSPutChangeAndVal function combines this functionality and is the preferred function for changes that require an attribute value.

NCP Calls

None

See Also

NWDSPutAttrVal (page 312), NWDSModifyObject (page 286)

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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPutChangeAndVal  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   nuint32             changeType,
   pnstr8              attrName,
   nuint32             syntaxID,
   nptr                attrVal);
```

Pascal

```
uses netwin32

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 (see Section 5.5, "Change Types for Modifying Objects," on page 466).

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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 eDirectory object.

A change record includes the name of the attribute and the type of change to be performed.

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.

The buf parameter points to a Buf_T, which is allocated by NWDSAllocBuf and initialized by NWDSInitBuf.

The syntaxID parameter 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 "Attribute Syntax Definitions" (*NDK: Novell eDirectory Schema Reference*).

The attrVal parameter 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 "Attribute Syntax Definitions" to determine the data type associated with an attribute.

NCP Calls

None

See Also

NWDSPutAttrVal (page 312), NWDSPutChange (page 314), NWDSModifyObject (page 286)

NWDSPutClassItem

Stores a class name or attribute name in a request buffer to be used by an eDirectory schema function.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPutClassItem  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              itemName);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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” on page 67 for the complete steps to fill out a buffer to be used for defining a new class.

NCP Calls

None

See Also

NWDSReadClassDef (page 333), NWDSPutClassName (page 321), NWDSPutSyntaxName (page 325)

NWDSPutClassName

Stores a class name in a request buffer to be used by an eDirectory function.

NetWare Server: 4.x, 5.x, 6.x

Platform: Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPutClassName  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              itemName);
```

Pascal

```
uses netwin32

Function NWDSPutClassName
  (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 being prepared.

itemName

(IN) Points to the class name to be stored in the request buffer.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 (page 333), NWDSPutClassItem (page 319)

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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsfilt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPutFilter  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pFilter_Cursor_T    cur,
   void                (N_FAR N_CDECL  *freeVal)(
                            nuint32   syntax,
                            nptr      val) );
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The buf parameter points to a Buf_T, which is allocated by NWDSAllocBuf and initialized for a DSV_SEARCH_FILTER 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.

The freeVal parameter points to a function freeing the attribute values. The function is passed the syntax attribute ID and the address of the area to free. The freeVal parameter can be NULL, in which case no attribute values are freed.

NCP Calls

None

See Also

NWDSAddFilterToken (page 84), NWDSAllocFilter (page 97), NWDSDelFilterToken (page 140), NWDSFreeFilter (page 162)

NWDSPutSyntaxName

Stores a syntax name in a request buffer to be used by a eDirectory function.

NetWare Server: 4.x, 5.x, 6.x

Platform: Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsbuft.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSPutSyntaxName  (
   NWDSContextHandle   context,
   pBuf_T              buf,
   pnstr8              itemName);
```

Pascal

```
uses netwin32

Function NWDSPutSyntaxName
  (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 being prepared.

itemName

(IN) Points to the syntax name to be stored in the request buffer.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSPutSyntaxName is a macro that calls NWDSPutClassItem.

The buf parameter points to Buf_T, which is allocated by NWDSAllocBuf and initialized by NWDSInitBuf.

NCP Calls

None

See Also

NWDSReadClassDef (page 333), NWDSPutClassItem (page 319)

NWDSRead

Reads values from one or more of the specified object's attributes.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSRead  (
   NWDSContextHandle   context,
   pnstr8              object,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint_ptr           iterationHandle,
   pBuf_T              objectInfo);
```

Pascal

```
uses netwin32

Function NWDSRead
  (context : NWDSContextHandle;
   object : pnstr8;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint_ptr;
   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 (see Section 5.16, "Information Types for Search and Read," on page 476).

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

The infoType, allAttrs, and attrNames parameters indicate what attribute information is requested.

If NWDSRead is called using infoTypes DS_VALUE_INFO (3) or DS_ABBREVIATED_VALUE (4), eDirectory returns the attribute value flags and modification timestamp. If the attribute value has been deleted, but not synchronized, eDirectory can return a value flag of DS_NOT_PRESENT (value not present) and no data. In this case, the attribute value length is zero. NWDSComputeAttrValSize returns the correct size of 0, and the application should not call NWDSGetAttrVal to retrieve the data because there is no data to retrieve.

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. Initialize the buffer with a value of DSV_READ. For more information on setting up this buffer, see "Reading Attributes of eDirectory Objects" on page 62.

The iterationHandle parameter controls retrieval of list results larger than the result buffer pointed to by attrNames.

Before the initial call to NWDSRead, set the iterationHandle parameter 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, such as NWDSRead, might take a long time to complete. For example, listing all of the User objects on a corporate network might be too time consuming. Developers should use NWDSCloseIteration to allow users of their applications to abort an iterative process that is taking too long to complete.

To end the read operation before complete results have been retrieved, call NWDSCloseIteration with a value of DSV_READ to free memory and states associated with the read operation.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSReadObjectInfo (page 340)

NWDSReadAttrDef

Retrieves information about eDirectory schema attribute definitions.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdssch.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSReadAttrDef  (
   NWDSContextHandle   context,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint_ptr           iterationHandle,
   pBuf_T              attrDefs);
```

Pascal

```
uses netwin32

Function NWDSReadAttrDef
  (context : NWDSContextHandle;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint_ptr;
   attrDefs : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

infoType

(IN) Specifies the information type desired (see Section 5.14, “Information Types for Attribute Definitions,” on page 475).

allAttrs

(IN) Specifies the scope of the request: TRUE=information concerning all attributes is requested; FALSE=only attributes named in attrNames are requested.

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The infoType, allAttrs, and attrNames parameters indicate what eDirectory Schema attribute information is requested.

If allAttrs is TRUE, information about all attributes in the eDirectory schema is requested. In this case, attrNames is ignored and can be set to NULL. If allAttrs is FALSE, only the attributes specified by attrNames are requested.

The iterationHandle parameter 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 the result buffer holds the complete results when NWDSReadAttrDef 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 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.

The attrDefs parameter points to a request buffer containing the requested attribute information. This buffer contains either a list of attribute names, or a sequence of attribute names and definitions depending upon the value of infoType mentioned above.

For step-by-step instructions, see “Reading an Attribute Definition” on page 72.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSReadClassDef (page 333)

NWDSReadClassDef

Retrieves information about eDirectory schema object class definitions.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdssch.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSReadClassDef  (
   NWDSContextHandle   context,
   nuint32             infoType,
   nbool8              allClasses,
   pBuf_T              classNames,
   pnint_ptr           iterationHandle,
   pBuf_T              classDefs);
```

Pascal

```
uses netwin32

Function NWDSReadClassDef
  (context : NWDSContextHandle;
   infoType : nuint32;
   allClasses : nbool8;
   classNames : pBuf_T;
   iterationHandle : pnint_ptr;
   classDefs : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

infoType

(IN) Specifies the information type desired (see Section 5.15, “Information Types for Class Definitions,” on page 476).

allClasses

(IN) Specifies whether information for every object class should be returned: TRUE=all classes; FALSE=only the classes specified in classNames are requested.

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 information.

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The infoType, allClasses, and classNames parameter indicate the type of object class information requested.

If allClasses is TRUE, information about all classes in the eDirectory 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.

The iterationHandle parameter 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.

If the result buffer holds the complete results when NWDSReadClassDef returns from its initial call, 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_DEF to free memory and states associated with NWDSReadClassDef.

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.

The classDefs parameter 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.

For step by step instructions, see “Reading a Class Definition” on page 70.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSReadAttrDef (page 330)

NWDSReadNDSInfo

Reads NDSPING information into a buffer for retrieval.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSReadNDSInfo  (
   NWCONN_HANDLE   connHandle,
   nflag32         requestedFields,
   pBuf_T          resultBuffer);
```

Pascal

```
uses netwin32

Function NWDSReadNDSInfo
  (connHandle : NWCONN_HANDLE;
   requestedFields : nflag32;
   resultBuffer : pBuf_T;
) : NWDSCCODE;
```

Parameters

connHandle

(IN) Specifies a connection handle to the eDirectory server.

requestedFields

(IN) Specifies the DSPING flags that control the information returned (see Section 5.19, “eDirectory Ping Flags,” on page 479).

resultBuffer

(OUT) Points to the buffer receiving the requested data.

Return Values

These are common return values.

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWDSReadNDSInfo places fields of information designated by Section 5.19, “eDirectory Ping Flags,” on page 479 into a reply buffer. These fields can then be retrieved one at a time with NWDSGetNDSInfo (page 208).

For instructions on how to use NWDSReadNDSInfo, see “Accessing eDirectory Ping Information” on page 54.

NCP Calls

0x2222 104 01 Ping for eDirectory NCP

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSGetNDSInfo (page 208), NWGetNLMInfo (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/srvr_enu/data/sdk354.html)

NWDSReadObjectDSIInfo

Returns the DSI object information not stored in the attributes of an object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSReadObjectDSIInfo  (
   NWDSContextHandle   context,
   pnstr8              object,
   nuint32             infoLength,
   nptr                objectInfo);
```

Pascal

```
uses netwin32

Function NWDSReadObjectDSIInfo  (
   context : NWDSContextHandle;
   object : 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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Object information can be useful to applications browsing the eDirectory 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 Section 5.11, “DCK_DSI_FLAGS Values,” on page 472. The default settings return the object's entry flags, subordinate count, modification time, base class, RDN, and DN.

To read the information in the objectInfo buffer, use the NWDSGetDSIInfo function.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSGetDSIInfo (page 199), NWDSGetObjectNameAndInfo (page 217), NWDSRead (page 327), NWDSReadObjectInfo (page 340)

NWDSReadObjectInfo

Returns object information not stored in the attributes of the object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSReadObjectInfo  (
   NWDSContextHandle   context,
   pnstr8              object,
   pnstr8              distinguishedName,
   pObject_Info_T      objectInfo);
```

Pascal

```
uses netwin32

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, which may be distinguished or relative depending upon the context flags.

objectInfo

(OUT) Points to the non-attribute information about the object.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

This function returns the name of the base class used to create the object, entry flags, modification time, and the number of objects that are subordinate to this object. This information can be useful to applications browsing the eDirectory tree. The flags set on the context handle influence the information returned. For more information, see Section 5.6, “Context Keys and Flags,” on page 467.

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 Unicode characters (Unicode characters are always 16 bits). 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRead (page 327)

NWDSReadReferences

Returns information about the references of the specified object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSReadReferences  (
   NWDSContextHandle   context,
   pnstr8              serverName,
   pnstr8              objectName,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   uint32              timeFilter,
   pnint_ptr           iterationHandle,
   pBuf_T              objectInfo);
```

Pascal

```
uses netwin32

Function NWDSReadReferences
  (context : NWDSContextHandle;
   serverName : pnstr8;
   objectName : pnstr8;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   timeFilter : nuint32;
   iterationHandle : pnint_ptr;
   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 reference information is to be read.

infoType

(IN) Specifies the type of information to be returned (see Section 5.16, “Information Types for Search and Read,” on page 476).

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.

0x0000 0000	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
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The NWDSReadReference function returns information about objects that are referencing a specified object on a specified server. The objectName parameter specifies the object, for example object X, that exists on the server. The function returns information about the objects, for example objects Y and Z, that are listed in the Reference attribute of object X. Information about each object listed in the Reference attribute is returned in the objectInfo buffer.

This function reads the Reference attribute of the object. Entires are added to this attribute when the object is referenced in an ACL of another object or the object is added to a list attribute. For example, User objects can belong to Group objects which have a Member attribute. When a User is added to just an attribute, the Group is added to the User's Reference attribute.

The infoType, allAttrs, and attrNames parameters indicate what attribute information to return concerning the reference entries.

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, and infoType is not meaningful. In this case, the return value of NWDSReadReferences 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 attributes to be returned. For more information, see “Preparing eDirectory Output Buffers” on page 53.

The information in the result buffer depends on infoType. To read the information in the buffer, first call NWDSGetObjectCount. For more information, see “Retrieving Results from eDirectory Output Buffers” on page 53.

The iterationHandle parameter controls retrieval of read results larger than the result buffer pointed to by objectInfo. Before the initial call to NWDSReadReferences, set the iterationHandle parameter to NO_MORE_ITERATIONS.

If the result buffer holds the complete results when NWDSReadReferences 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 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 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 infoType is set to 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 and then all of the values associated with the attribute. The next attribute name and its values can then be removed. 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSReadObjectInfo (page 340)

NWDSReadSyntaxDef

Returns the syntax definition for a given eDirectory syntax identifier.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSReadSyntaxDef  (
   NWDSContextHandle   context,
   nuint32             syntaxID,
   pSyntax_Info_T      syntaxDef);
```

Pascal

```
uses netwin32

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 (see Section 5.26, “Syntax IDs,” on page 487).

syntaxDef

(OUT) Points to a Syntax_Info_T structure, which receives the syntax definition.

Return Values

0x0000 0000	SUCCESSFUL

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

The syntaxID parameter is an identifier (not a string) of a syntax. These identifiers (such as SYN_TEL_NUMBER) are enumerated in the nwdsdefs.h file.

This is a local function. Syntaxes are well known.

NCP Calls

None

NWDSReadSyntaxes

Enumerates syntax definitions or retrieves specific eDirectory schema syntax definitions.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSReadSyntaxes  (
   NWDSContextHandle   context,
   nuint32             infoType,
   nbool8              allSyntaxes,
   pBuf_T              syntaxNames,
   pnint_ptr           iterationHandle,
  pBuf_T               syntaxDefs);
```

Pascal

```
uses netwin32

Function NWDSReadSyntaxes
  (context : NWDSContextHandle;
   infoType : nuint32;
   allSyntaxes : nbool8;
   syntaxNames : pBuf_T;
   iterationHandle : pnint_ptr;
   syntaxDefs : pBuf_T
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

infoType

(IN) Specifies the type of information desired: DS_SYNTAX_NAMES equals names only, and DS_SYNTAX_DEFS equals names, IDs, and matching rules.

allSyntaxes

(IN) Specifies the scope of the request: TRUE=all syntaxes and FALSE= selected syntaxes specified by syntaxNames.

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

The infoType, allSyntaxes, nd syntaxNames parameters indicate what syntax information is requested.

The infoType parameter uses two flags which specify what information is returned.

Flag	Description
DS_SYNTAX_NAMES	Returns only syntax names
DS_SYNTAX_DEFS	Returns syntax names with a Syntax_Info_T (page 457) structure that contains the syntax ID and the syntax matching rules

If allSyntaxes is TRUE, information is returned for all syntaxes defined for the eDirectory tree, 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, and infoType is not meaningful.

The syntaxNames parameter 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.

The iterationHandle parameter 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.

If the result buffer holds the complete results when NWDSReadSyntaxes 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 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 definition.

The syntaxDefs parameter 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 definitions, 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" on page 73.

NCP Calls

None

NWDSReloadDS

Requests a specified server to unload and then load the DS NLM.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSReloadDS  (
   NWDSContextHandle   context,
   pnstr8              serverName);
```

Pascal

```
uses netwin32

Function NWDSReloadDS
  (context : NWDSContextHandle;
   serverName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

serverName

(IN) Points to the server to send the request to.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsname.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSRemoveAllTypes  (
   NWDSContextHandle   context,
   pnstr8              name,
   pnstr8              typelessName);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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. For more information, see the DCV_TYPELESS_NAMES key in “DCK_FLAGS Key” on page 18.

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 Unicode characters (Unicode characters are always 16 bits). 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 eDirectory schema.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdssch.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSRemoveAttrDef  (
   NWDSContextHandle   context,
   pnstr8              attrName);
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

The attrName parameter identifies the attribute definition to be deleted from the schema.

NOTE: Clients cannot subtract from the standard set of attribute definitions defined by the eDirectory operational schema (these attributes are flagged nonremovable). Clients can, however, add and remove non-standard definitions (if not in use).

If an attribute has been added to an object class, the object class must be deleted before the attribute definition can be deleted.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRemoveClassDef (page 357)

NWDSRemoveClassDef

Deletes a class definition from the eDirectory schema.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdssch.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSRemoveClassDef  (
   NWDSContextHandle   context,
   pnstr8              className);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Calling NWDSRemoveClassDef is not allowed if the class is referenced by any other class, or if objects of this class exist in the eDirectory database.

The className parameter identifies the class whose definition is to be removed.

NOTE: Clients cannot subtract from the standard set of class definitions defined by the eDirectory operational 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRemoveAttrDef (page 355)

NWDSRemoveObject

Removes a leaf object (either an object or an alias) from the eDirectory tree.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSRemoveObject  (
   NWDSContextHandle   context,
   pnstr8              object);
```

Pascal

```
uses netwin32

Function NWDSRemoveObject
  (context : NWDSContextHandle;
   object : 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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

NWDSRemovePartition

Removes an existing partition from eDirectory by deleting its master replica.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSRemovePartition  (
   NWDSContextHandle   context,
   pnstr8              partitionRoot);
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

The partitionRoot parameter 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRemoveReplica (page 363)

NWDSRemoveReplica

Removes a replica from the replica set of an eDirectory partition.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSRemoveReplica  (
   NWDSContextHandle   context,
   pnstr8              server,
   pnstr8              partitionRoot);
```

Pascal

```
uses netwin32

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 eDirectory partition whose replica is being deleted.

Return Values

These are common return values.

0x0000 0000	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 eDirectory
0x89FE	BAD_PACKET
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSRemovePartition (page 361)

NWDSRemSecurityEquiv

Removes a security equivalence from the specified object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSRemSecurityEquiv  (
   NWDSContextHandle   context,
   pnstr8              equalFrom,
   pnstr8              equalTo);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 caller of the function does not have sufficient rights to the object specified by equalFrom, 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAddSecurityEquiv (page 93)

NWDSRepairTimeStamps

Sets the time stamps for all of a partition's objects and their attributes to the current time on the NetWare server where the master replica is located.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSRepairTimeStamps  (
   NWDSContextHandle   context,
   pnstr8              partitionRoot);
```

Pascal

```
uses netwin32

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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, eDirectory will synchronize all replicas to match the information in the master replica.

IMPORTANT: 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 eDirectory Event Notification Services. After NWDSRepairTimeStamps is called, eDirectory 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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

NWDSReplaceAttrNameAbbrev

Replaces the abbreviated attribute name with its unabbreviated name.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSReplaceAttrNameAbbrev  (
   NWDSContextHandle   context,
   pnstr8              inStr,
   pnstr8              outStr);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 Unicode characters (Unicode characters are always 16 bits). 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

NWDSResolveName

Returns a connection handle and an object ID for the object name.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsname.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSResolveName  (
   NWDSContextHandle     context,
   pnstr8                objectName,
   NWCONN_HANDLE N_FAR  *conn,
   pnuint32              objectID);
```

Pascal

```
uses netwin32


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 eDirectory object ID.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAddObject (page 87), NWDSAuditGetObjectID (obsolete 06/03) (page 99)

NWDSRestoreObject

Restores an object's attribute names and values that were saved by calling NWDSBackupObject.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

NN_EXTERN_LIBRARY (NWDSCCODE)  NWDSRestoreObject  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   pnint_ptr           iterationHandle,
   nbool8              more,
   nuint32             size,
   pnuint8             objectInfo);
```

Pascal

```
uses netwin32

Function NWDSRestoreObject
  (context : NWDSContextHandle;
   objectName : pnstr8;
   iterationHandle : pnint_ptr;
   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.

iterationHandle

(IN) Points to value that eDirectory uses internally when repeative calls must be made to restore an object. Initialize to -1.

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.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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.

The iterationHandle parameter is used differently in this function. In other functions, the amount of data returned from the server is potentially larger than a single eDirectory message can accommodate. Here, the opposite is true. The request can be larger than the largest eDirectory message. When the more parameter is set to TRUE, the client indicates that not all of the object data is in the current request and that more data is coming. When the client sets the more parameter to FALSE, the client indicates the completion of a series of restore requests. Only on completition does eDirectory process the request.

In the initial call to NWDSRestoreObject, set the iterationHandle parameter to NO_MORE_ITERATIONS. On subsequent requests, set it to the value returned in the preceding reply.

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.

To abort the restoration of an object before sending all the information about an object, call NWDSCloseIteration with an operation type of DSV_RESTORE_ENTRY.

The size parameter 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 eDirectory NCP

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSBackupObject (page 107)

NWDSReturnBlockOfAvailableTrees

Scans the bindery of the specified connection and returns matching tree objects.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#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

```
uses netwin32

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 or NULL for the preferred tree.

connHandle

(IN) Specifies the connection handle to be used in scanning for eDirectory 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 (optional).

maxTreeNames

(IN) Specifies the 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) Points to the actual number of tree names that were returned in the arrayOfNames parameter.

totalUniqueTrees

(OUT) Points to the total number of tree names found that match the scan criteria (might be greater than the numberOfTrees parameter since the buffer size controls how many names are actually returned).

Return Values

These are common return values.

0x0000 0000	SUCCESSFUL
	NO_SUCH_OBJECT
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

This function returns tree names in sorted order and with duplicates removed.

To set up a scan, place a filter string in the scanFilter parameter and set the endBoundString parameter to NULL. The filter string can contain a wildcard, such as "nov*". The results of this scan would include all tree names that begin with "nov".

If you want to set an end boundary for the scan, place a second filter string in the endBoundString parameter. For example, if the scanFilter parameter contains "a*" and endBoundString 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. You are responsible for setting up and initializing an output buffer before calling NWDSReturnBlockOfAvailableTrees. You 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,,);
}
```

You could set the maxTreeNames parameter equal to 0 and call NWDSReturnBlockOfAvailableTrees. The value returned in the totalUniqueTrees parameter could then be used to determine how much memory to allocate for the arrayOfNames parameter.

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. The DCV_XLATE_STRINGS flag determines whether local code page format or Unicode strings are returned. For more information, see "DCK_FLAGS Key" on page 18.

Continuing a Scan. If the value returned in the totalUniqueTrees parameter is greater than the value returned in the numberOfTrees parameter, there are more tree names that meet the scan filter criteria than were returned in the arrayOfNames parameter. If this is the case, 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 the scanFilter and endBoundString parameters the same as before, and place the name that was returned in the last element of the arrayOfNames parameter into the lastBlocksString parameter.

NCP Calls

0x2222 23 55 Get Server Sources Information

See Also

NWDSScanConnsForTrees (page 379), NWDSScanForAvailableTrees (page 381)

NWDSScanConnsForTrees

Scans existing connections for tree names.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSScanConnsForTrees  (
   NWDSContextHandle   context,
   nuint               numOfPtrs,
   pnuint              numOfTrees,
   ppnstr8             treeBufPtrs);
```

Pascal

```
#include <nwdsconn.inc>

Function NWDSScanConnsForTrees
  (context : NWDSContextHandle;
   numOfPtrs : nuint;
   numOfTrees : pnuint;
   Var treeBufPtrs : ppnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request or NULL for the preferred tree.

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory until authentication.

The context parameter is used to determine the character type for the tree name. The DCV_XLATE_STRINGS flag determines whether local code page format or Unicode strings are returned. For more information, see “DCK_FLAGS Key” on page 18

NOTE: When NWDSScanConnsForTrees is called on a workstation running Client32, which runs on Windows 95, it does not actually scan connections. This call utilizes Client32’s ability to scan for identities.

NCP Calls

None

See Also

NWDSScanForAvailableTrees (page 381), NWDSReturnBlockOfAvailableTrees (page 376)

NWDSScanForAvailableTrees

Scans a connection for tree objects.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSScanForAvailableTrees  (
   NWDSContextHandle   context,
   NWCONN_HANDLE       connHandle,
   pnstr               scanFilter,
   pnint32             scanIndex,
   pnstr               treeName);
```

Pascal

```
uses netwin32

Function NWDSScanForAvailableTrees
  (context : NWDSContextHandle;
   connHandle : NWCONN_HANDLE;
   scanFilter : pnstr;
   Var scanIndex : pnint32;
   treeName : pnstr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request or NULL for the preferred tree.

connHandle

(IN) Specifies the connection handle to be used in scanning for eDirectory 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 0000	SUCCESSFUL
0x89FC	BIND_NO_SUCH_OBJECT
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

Remarks

NWDSScanForAvailableTrees uses the connection specified in connHandle to scan for eDirectory tree objects (object type 0x7802) using the server bindery (the dynamic bindery is used for NetWare 4.x and 5.x servers). When the list of tree objects is exhausted, the NWDSScanForAvailableTrees function returns BIND_NO_SUCH_OBJECT.

This function may return duplicate tree names because it returns the tree name from each server which is advertising it. To receive a sorted list with duplicates removed, use NWDSReturnBlockOfAvailableTrees (page 376).

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 eDirectory 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. The DCV_XLATE_STRINGS flag determines whether local code page format or Unicode strings are returned. For more information, see "DCK_FLAGS Key" on page 18

NCP Calls

0x2222 23 55 Scan Bindery Object

See Also

NWDSScanConnsForTrees (page 379), NWDSReturnBlockOfAvailableTrees (page 376)

NWDSSearch

Searches a branch of the eDirectory tree for objects satisfying a specified set of requirements.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSSearch  (
   NWDSContextHandle   context,
   pnstr8              baseObjectName,
   nint                scope,
   nbool8              searchAliases,
   pBuf_T              filter,
   nuint32             infoType,
   nbool8              allAttrs,
   pBuf_T              attrNames,
   pnint_ptr           iterationHandle,
   nint32              countObjectsToSearch,
   pnint32             countObjectsSearched,
   pBuf_T              objectInfo);
```

Pascal

```
uses netwin32

Function NWDSSearch
  (context : NWDSContextHandle;
   baseObjectName : pnstr8;
   scope : nint;
   searchAliases : nbool8;
   filter : pBuf_T;
   infoType : nuint32;
   allAttrs : nbool8;
   attrNames : pBuf_T;
   iterationHandle : pnint_ptr;
   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 (see Section 5.22, “Scope Flags,” on page 483).

searchAliases

(IN) Specifies whether to dereference subordinate aliases in the search:

TRUE Aliases will be dereferenced

FALSE Aliases will not be dereferenced

filter

(IN) Points to a search filter constructed by calling the NWDSAddFilterToken function. This parameter must be specified (cannot be NULL). To specify no filtering (return all values) construct a filter equivalent to objectclass=*.

infoType

(IN) Specifies the type of information to return (see Section 5.16, “Information Types for Search and Read,” on page 476).

allAttrs

(IN) Specifies the scope of the information to return:

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) Reserved for future use.

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

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 tries an alternate replica for the partition. If none of the alternate replicas can be contacted, 0 results are returned without an accompanying error. Please confirm that your client can contact all servers in the tree.

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.

Aliases are dereferenced while locating the base object unless the context flag associated with DCV_DEREF_ALIASES is not set. For more information, see Section 5.6, “Context Keys and Flags,” on page 467.

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. For information on building a search filter, see Section 1.4, “Search Requests,” on page 30. For step-by-step instructions, see “Searching eDirectory” on page 63.

The infoType, allAttrs, nd 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.

NOTE: On large networks, iterative processes, such as 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. Developers should call the NWDSCloseIteration (page 126) function to allow users of their applications to abort an iterative process that is taking too long to complete.

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 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 example code, see ndssearc.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/index.htm).

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

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSCloseIteration (page 126), NWDSAddFilterToken (page 84), NWDSAllocFilter (page 97), NWDSDelFilterToken (page 140), NWDSFreeFilter (page 162), NWDSPutFilter (page 323)

NWDSSetContext

Sets the information in an NDS context handle.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSSetContext  (
   NWDSContextHandle   context,
   nint                key,
   nptr                value);
```

Pascal

```
uses netwin32

Function NWDSSetContext
  (context : NWDSContextHandle;
   key : nint;
   value : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the context handle for the request.

key

(IN) Specifies the information to set (see Section 5.6, “Context Keys and Flags,” on page 467).

value

(IN) Points to the values to use in changing the context handle information.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Applications cannot directly change context handle information; they must use the NWDSSetContext function to change information.

The key parameter specifies the type of information to change, and the value parameter points to the new value. The value parameter must point to a variable matching the data type specified by the key parameter. For data types and defined keys, see Section 5.6, “Context Keys and Flags,” on page 467.

The NWDSSetContext function must be called for each key that has information that needs to be modified.

NCP Calls

None

See Also

NWDSCreateContextHandle (page 133), NWDSFreeContext (page 160), NWDSGetContext (page 191)

NWDSSetCurrentUser

Sets the user handle of an eDirectory user.

Local Servers: nonblocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: NDS

Syntax

```
#include <nwconn.h>
#include <nwdsapi.h>

 int NWDSSetCurrentUser (
   int   userHandle);
```

Parameters

userHandle

(IN) Specifies the user handle to set as the current user's handle.

Return Values

0	(0x00)	SUCCESSFUL
2	(0x02)	INVALID_USER

Remarks

This function sets the value of the current user in the Thread Group Control Structure (TGCS). The current user determines which eDirectory authentication information is used. For more information, see “Establishing Identities to Multiple eDirectory Trees—NLM Platform” on page 55.

See Also

NWDSGetCurrentUser (page 196)

NWDSSetDefNameContext

Sets the default name context for a specified tree.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsconn.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSSetDefNameContext  (
   NWDSContextHandle   context,
   nuint               nameContextLen
   pnstr8              nameContext);
```

Pascal

```
#include <nwdsconn.inc>

Function NWDSSetDefNameContext
  (context : NWDSContextHandle;
   nameContextLen : nuint;
   nameContext : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request or NULL for the preferred tree.

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 return only 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 only 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 eDirectory 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 (page 410), NWDSGetDefNameContext (page 197), NWSetDefaultNameContext (page 427)

NWDSSetMonitoredConnection (obsolete 06/03)

Tracks the connection, but is now obsolete.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwndscon.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSSetMonitoredConnection
  (NWCONN_HANDLE   connHandle);
```

Pascal

```
uses netwin32

Function NWDSSetMonitoredConnection
  (connHandle : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the connection handle of the desired connection.

Return Values

These are common return values; see "NDS Return Values" for more information.

0x0000 0000	SUCCESSFUL
nonzero value	UNSUCCESSFUL

Remarks

When a user logs in to the eDirectory tree, 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 eDirectory functions such as NWDSWhoAmI do not return valid information.

No replacement is needed for this function as monitored connections are managed autmoatically by the client software.

NCP Calls

None

See Also

NWDSOpenMonitoredConn (page 299)

NWDSSplitPartition

Divides a partition into two partitions at a specified object.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSSplitPartition  (
   NWDSContextHandle   context,
   pnstr8              subordinatePartition,
   nflag32             flags);
```

Pascal

```
uses netwin32

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 where the partition will be split and which will become the root of the subordinate partition.

flags

(IN) Reserved; pass in 0.

Return Values

These are common return values.

0x0000 0000	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 eDirectory
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

Operations to split a partition are always performed on the master replica. If the context handle points to a read-write or read-only replica, the request is redirected to the master replica.

The object specified becomes the root object of the subordinate partition.

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSAddReplica (page 91), NWDSJoinPartitions (page 246)

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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdspart.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSSyncPartition  (
   NWDSContextHandle   context,
   pnstr8              server,
   pnstr8              partition,
   nuint32             seconds);
```

Pascal

```
uses netwin32

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 before beginning the synchronization process.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 eDirectory NCP
0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSSyncReplicaToServer (page 398), NWDSSyncSchema (page 400)

NWDSSyncReplicaToServer

Requests a replica to synchronize with a specific server.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSSyncReplicaToServer  (
   NWDSContextHandle   context,
   pnstr8              serverName,
   pnstr8              partitionRootName,
   pnstr8              destServerName,
   nuint32             actionFlags,
   nuint32             delaySeconds);
```

Pascal

```
uses netwin32

Function NWDSSyncReplicaToServer
  (context : NWDSContextHandle;
   serverName : pnstr8;
   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 to be synchronized.

partitionRootName

(IN) Points to the name of the partition whose replica is to be synchronized.

destServerName

(IN) Points to the server to which the replica should synchronize.

actionFlags

(IN) Specifies the synchronization action to be taken.

delaySeconds

(IN) Specifies the number of seconds to delay before beginning the synchronization.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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. Ignored unless SF_DO_IMMEDIATE is also set

SF_SEND_ALL

Synchronize all objects, as opposed to only those that have changed.

SF_SEND_SINGLE_ENTRY

Synchronize only the subject object. Ignored if SF_SEND_ALL is set.

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 (page 396)

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, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdssch.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSSyncSchema  (
   NWDSContextHandle   context,
   pnstr8              server,
   nuint32             seconds);
```

Pascal

```
uses netwin32

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 before beginning the synchronization.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

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 (page 396)

NWDSUnlockConnection (obsolete 06/03)

Enables the connection to be placed on the LRU list and unlicenses the connection if no other resources are allocated, but is now obsolete.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwndscon.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSUnlockConnection
  (NWCONN_HANDLE   connHandle);
```

Pascal

```
uses netwin32

Function NWDSUnlockConnection
  (connHandle : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the connection to unlock.

Return Values

These are common return values; see "NDS Return Values" for more information.

0x0000 0000 SUCCESSFUL

Remarks

If there are no other tasks having resources 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.

Use NWCCUnlincenseConn in place of this function.

NCP Calls

None

See Also

NWCCGetConnRefInfo (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk589.html), NWCCLicenseConn (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk625.html), NWDSAuthenticate (obsolete 06/03) (page 101)

NWDSVerifyObjectPassword

Verifies the password of an object. Does not support international or extended characters in passwords.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSVerifyObjectPassword  (
   NWDSContextHandle   context,
   nflag32             optionsFlag,
   pnstr8              objectName,
   pnstr8              password);
```

Pascal

```
uses netwin32

Function NWDSVerifyObjectPassword
  (context : NWDSContextHandle;
   optionsFlag : nflag32;
   objectName : pnstr8;
   password : pnstr8
) : NWDSCCODE;
```

Parameters

context

(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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (–001 to –799).

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.

NOTE: NWDSVerifyPwdEx (page 406) supports international and extended characters and is recommended in place of NWDSVerifyObjectPassword.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSGenerateObjectKeyPair (page 167), NWDSLogin (page 270), NWDSChangeObjectPassword (page 116)

NWDSVerifyPwdEx

Verifies the password of an object. Supports international and extended characters in passwords. NWDSVerifyPwdEx was not implemented in the old NLMs so you might not find it if you are using an old netnlm32.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsasa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSVerifyPwdEx  (
   NWDSContextHandle   context,
   pnstr8              objectName,
   nuint32             pwdFormat,
   nptr                pwd);
```

Pascal

```
uses netwin32

Function NWDSVerifyPwdEx
  (context : NWDSContextHandle;
   objectName : pnstr8;
   pwdFormat : nuint32;
   pwd : nptr
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the handle to the name context structure.

objectName

(IN) Points to the object name (under the context) of the object to verify.

pwdFormat

(IN) Specifies the format of the password data. Select from the following:

Password Format	Description
PWD_UNICODE_STRING	Allows any unicode string to be used as a password.
PWD_UTF8_STRING	Allows any UTF8 string to be used as a password.
PWD_RAW_C_STRING	Allows any arbitrary NULL-terminated data to be used as a password. Passwords specified with this format are not interoperable with unicode and UTF8 passwords.

pwd

(IN) Points to the password for the object.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values”.

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.

NOTE: The PWD_RAW_C_STRING password format allows any arbitrary NULL-terminiated data to be used as a password. Passwords specified with this format are not interoperable with unicode and UTF8 passwords.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSGenerateKeyPairEx (page 164), NWDSLoginEx (page 272), NWDSChangePwdEx (page 119)

NWDSWhoAmI

Returns the name of the object currently logged in to eDirectory.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsdsa.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSWhoAmI  (
   NWDSContextHandle   context,
   pnstr8              objectName);
```

Pascal

```
uses netwin32

Function NWDSWhoAmI
  (context : NWDSContextHandle;
   objectName : pnstr8
) : NWDSCCODE;
```

Parameters

context

(IN) Specifies the NDS context for the request.

objectName

(OUT) Points to the name of the object logged in to eDirectory.

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

If the object is not currently logged in, NWDSWhoAmI returns an error.

The object name is returned in partial dot form. Whether the name in objectName is returned as a full name or a partial name depends upon the setting of the context flags:

- 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 distinguished 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 Unicode characters (Unicode characters are always 16 bits). One character is used for NULL termination.

NCP Calls

None

NWGetDefaultNameContext

Allows the user to get the default name context.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h
or
#include <nwndscon.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWGetDefaultNameContext  (
   nuint16       bufferSize,
   pnuint8       context);
```

Pascal

```
uses netwin32

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 "NDS Return Values" for more information.

0x0000 0000	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.

NCP Calls

None

See Also

NWSetDefaultNameContext (page 427)

NWGetFileServerUTCTime

Returns the Coordinated Universal Time (UTC) setting of a server.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWGetFileServerUTCTime  (
   NWCONN_HANDLE   conn,
   pnuint32        time);
```

Pascal

```
uses netwin32

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.

0x0000 0000	SUCCESSFUL
0xFD6D	ERR_TIME_NOT_SYNCHRONIZED

nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWGetFileServerUTCTime determines the time setting on remote and local servers. (The time (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/prog_enu/data/sdk1823.html) (Program Management (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/prog_enu/data/h9qu926c.html)) function returns only 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

NWGetNumConnections

Returns the number of connections that can be supported by VLM.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwndscon.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWGetNumConnections  (
   pnuint16    numConnections);
```

Pascal

```
uses netwin32

Function NWGetNumConnections
  (numConnections : pnuint16
) : NWCCODE;
```

Parameters

numConnections

(OUT) Points to the number of connections supported.

Return Values

These are common return values; see "NDS Return Values" for more information.

0x0000 0000	SUCCESSFUL
0x8800	VLM_ERROR

Remarks

The number of connections can be configured in the net.cfg file.

NCP Calls

None

NWGetNWNetVersion

Returns the NWNet library version number.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#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

```
#uses netwin32

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 (page 201)

NWGetPreferredConnName

Gets the name of the preferred connection.

NetWare Server: N/A

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwndscon.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWGetPreferredConnName  (
   pnuint8   preferredName,
   pnuint8   preferredType);
```

Pascal

```
uses netwin32

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 “NDS Return Values” for more information.

0x0000 0000 SUCCESSFUL

Remarks

NWGetPreferredConnName will work only 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 (page 429)

NWIsDSAuthenticated

Returns whether eDirectory has credentials for a background authentication in the current eDirectory tree. This function is obsolete. Call NWDSCanDSAuthenticate (page 112), which indicates if there is an authenticated identity for the tree, or NWDSScanConnsForTrees (page 379), which returns a list of trees that have authenticated identities, instead.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwndscon.h>

N_EXTERN_LIBRARY (NWCCODE)  NWIsDSAuthenticated  (
   void);
```

Pascal

```
uses netwin32

Function NWIsDSAuthenticated
    : NWCCODE;
```

Return Values

These are common return values; see "NDS Return Values" for more information.

0x0000 0001	Authenticated through eDirectory
0x0000 0000	Not authenticated through eDirectory

Remarks

On NetWare, NWIsDSAuthenticated finds the current thread identity and checks to see if it is authenticated. On Windows, it returns true if there is any NDS authenticated identity.

NCP Calls

None

See Also

NWCCGetConnRefInfo (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk589.html), NWCCScanConnRefs (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/sdk697.html)

NWIsDSServer

Checks presence or absence of eDirectory on the server.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWIsDSServer  (
   NWCONN_HANDLE   conn,
   pnstr8          treeName);
```

Pascal

```
uses netwin32

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 the server specified by conn is an eDirectory server.

Return Values

These are common return values; see "NDS Return Values" for more information.

0x0000 0001	eDirectory NCP is hooked and eDirectory is running
0x0000 0000	Not eDirectory

NCP Calls

0x2222 104 01 Ping for eDirectory NCP

NWNetInit

Does the initial setup that is necessary before calling any other eDirectory functions.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWNetInit  (
   nptr   in,
   nptr   out);
```

Pascal

```
uses netwin32

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 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWNetInit initializes the eDirectory library. Both parameters, in and out, should be NULL when NWNetInit is called.

NCP Calls

None

See Also

NWNetTerm (page 425)

NWNetTerm

Shuts down and cleans up after the eDirectory library.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwdsmisc.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWNetTerm  (
   nptr   reserved);
```

Pascal

```
uses netwin32

Function NWNetTerm  (
   reserved : nptr
) : NWDSCCODE;
```

Return Values

0x0000 0000	SUCCESSFUL
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (–001 to –799).

Remarks

NWNetTerm terminates the eDirectory library.

Under VLM, NWNetTerm has no effect and other eDirectory functions may be called after calling NWNetTerm.

Under NLM, Windows 95, Windows 98, Windows NT, Windows 2000, and Windows XP, NWNetTerm should be called last as it will shut down and clean up after eDirectory.

If, after calling NWNetTerm, you want to call other eDirectory functions, call NWNetInit before calling any other eDirectory functions.

NCP Calls

None

See Also

NWNetInit (page 423)

NWSetDefaultNameContext

Sets the default name context.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwndscon.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWSetDefaultNameContext  (
   nuint16   contextLength,
   pnuint8   context);
```

Pascal

```
uses netwin32

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 "NDS Return Values" for more information.

0x0000 0000	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.

NCP Calls

None

See Also

NWGetDefaultNameContext (page 410)

NWSetPreferredDSTree

Sets the preferred eDirectory tree name in the requester's tables.

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP

Library: Cross-Platform NDS (NET*.*)

Service: NDS

Syntax

C

```
#include <nwnet.h>
or
#include <nwndscon.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWSetPreferredDSTree  (
   nuint16   length,
   pnuint8   treeName);
```

Pascal

```
uses netwin32

Function NWSetPreferredDSTree
  (length : nuint16;
   treeName : pnuint8
) : NWCCODE;
```

Parameters

length

(IN) Specifies the length of the tree name.

treeName

(IN) Points to the eDirectory tree name.

Return Values

These are common return values; see "NDS Return Values" for more information.

0x0000 0000	SUCCESSFUL
0x8836	INVALID_PARAMETER

Remarks

NWSetPreferredDSTree sets a tree name for future eDirectory functions. The tree name may also be set in net.cfg. The maximum name length is 32 characters. If the tree name is too long, INVALID_PARAMETER is returned.

NCP Calls

None

Structures

4

This chapter lists alphabetically the structures used by the Novell® eDirectory™ functions and the eDirectory schema.

Asn1ID_T

Holds the ASN.1 ID of an object.

Service: NDS

Defined In: nwdsbuft.h and nwdsbuft.inc

Structure

C

```
typedef struct
{
   nuint32   length;
   nuint8    data [MAX_ASN1_NAME];
} Asn1ID_T;
```

Pascal

```
Asn1ID_T = Record
    length : nuint32;
    data : Array[0..MAX_ASN1_NAME-1] Of nuint8
 End;
```

Fields

length

Specifies the number of characters in the array.

data

Contains a BER encoded string which specifies the ASN.1 ID for the object class or attribute definition.

NDS 8 verifies that the ASN.1 is BER encoded. Previous versions of NDS do not verify the string.

Attr_Info_T

Contains information about an attribute definition.

Service: NDS

Defined In: nwdsbuft.h and nwdsbuft.inc

Structure

C

```
typedef struct
{
   nuint32    attrFlags ;
   nint32     attrSyntaxID ;
   nint32     attrLower ;
   nint32     attrUpper ;
   Asn1ID_T   asn1ID ;
} Attr_Info_T;
```

Pascal

```
 Attr_Info_T = Record
    attrFlags : nuint32;
    attrSyntaxID : nint32;
    attrLower : nint32;
    attrUpper : nint32;
    asn1ID : Asn1ID_T
  End;
```

Fields

attrFlags

Specifies the constraints assigned to the attribute (see Section 5.1, “Attribute Constraint Flags,” on page 461).

attrSyntaxID

Specifies the syntax ID of the attribute type (see Section 5.26, “Syntax IDs,” on page 487).

attrLower

Specifies the lower limit of the attribute.

attrUpper

Specifies the upper limit of the attribute.

asn1ID

Specifies 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. NetWare 5.x requires an ASN.1 identifier.

Back_Link_T

Contains eDirectory information for the attributes which use the Back Link syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   nuint32   remoteID ;
   pnstr8    objectName ;
} Back_Link_T;
```

Pascal

```
Back_Link_T = Record
      remoteID : nuint32;
      objectName : pnstr8
End;
```

Fields

remoteID

Identifies the reference that is valid on the server.

objectName

Identifies the server holding a reference.

Bit_String_T

Contains the optional bit string information of the Facsimile Telephone Number syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   nuint32   numOfBits ;
   pnuint8   data ;
} Bit_String_T;
```

Defined In

```
Bit_String_T = Record
      numOfBits : nuint32;
      data : pnuint8
 End;
```

Fields

numOfBits

Specifies the number of bits in the data that are used.

data

Points to the data, formatted according to Recommendation T.30.

Buf_T

Initializes and handles input and output buffers.

Service: NDS

Defined In: nwdsbuft.h and nwdsbuft.inc

Structure

C

```
typedef struct
{
   nuint32   operation ;
   nuint32   flags ;
   nuint32   maxLen ;
   nuint32   curLen ;
   pnuint8   lastCount ;
   pnuint8   curPos ;
   pnuint8   data ;
} Buf_T;
```

Pascal

```
Buf_T = Record
    operation : nuint32;
    flags : nuint32;
    maxLen : nuint32;
    curLen : nuint32;
    lastCount : pnuint8;
    curPos : pnuint8;
    data : pnuint8
 End;
```

Fields

operation

Specifies the verb of the function operating on the buffer; set by NWDSInitBuf. The operation determines the type of data in the buffer. For a list of operation types, see Section 5.3, “Buffer Operation Types and Related Functions,” on page 464.

flags

Specifies 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

Specifies 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

Specifies the length of the current buffer. Its value is 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

Points to the number of items in the data currently stored in the buffer. It is manipulated internally for iterative operations on the buffer.

curPos

Points to 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

Contains eDirectory information for the attributes that use the Case Ignore List syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct_ci_list
{
   struct_ci_list N_FAR   *next ;
   pnstr8                  s ;
} CI_List_T;
```

Pascal

```
CI_List_T = Record
      next : pCI_List_T;
      s : pnstr8
End;
```

Fields

next

Points to the next node containing a case-ignore string.

s

Points to a case-ignore string for this node.

Class_Info_T

Contains information about a object class definition.

Service: NDS

Defined In: nwdsbuft.h and nwdsbuft.inc

Structure

C

```
typedef struct
{
   nuint32    classFlags ;
   Asn1ID_T   asn1ID ;
} Class_Info_T;
```

Pascal

```
Class_Info_T = Record
    classFlags : nuint32;
    asn1ID : Asn1ID_T
 End;
```

Fields

classFlags

Specifies the type of object class (see Section 5.4, “Class Flags,” on page 465).

asn1ID

Specifies an optional object identifier encoded using the ASN.1-BER rules. If no ASN.1 object identifier has been registered for the attribute, a zero-length octet string is specified. NetWare 5.x requires an ASN.1 identifier.

EMail_Address_T

Contains the eDirectory information for the attributes that use the EMail Address syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   nuint32   type ;
   pnstr8    address ;
} EMail_Address_T;
```

Pascal

```
EMail_Address_T = Record
      type : nuint32;
      address : pnstr8
End;
```

Fields

type

Specifies the e-mail address type.

address

Points to the e-mail address, formatted according to the type field.

Remarks

The type field is specific to the e-mail application. MHS mail applications use the following types:

0	The data structure contains an e-mail address, in the form of non-MHS_Email_protcol:non-MHS_Email_Address (non_MHS_Email_Protocol is a 1-8 character string, and the non-MHS_Email_Address is a string for the actual address value)
1	The data structure contains an e-mail alias, in the form of non-MHS_Email_protcol:non-MHS_Email_Alias. (non_MHS_Email_Protocol is a 1-8 character string, and the non-MHS_Email_Alias is a string for the actual alias value)

The nwdsapi.h file defines a few e-mail address types in the EMAIL_ADDRESS_TYPE enumeration. However, e-mail applications can create new ones. eDirectory does not validate the types or the addresses; it stores the information. The e-mail application is responsible for ensuring that the information is stored in the syntax in a format that it can use.

Fax_Number_T

Contains the eDirectory information for the attributes that use the Facsimile Telephone Number syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   pnstr8         telephoneNumber ;
   Bit_String_T   parameters ;
} Fax_Number_T;
```

Pascal

```
Fax_Number_T = Record
      telephoneNumber : pnstr8;
      parameters : Bit_String_T
End;
```

Fields

telephoneNumber

Points to the next node containing a case-ignore string.

parameters

Specifies a case-ignore string for this node.

Filter_Cursor_T

Builds an expression tree to search for objects in eDirectory.

Service: NDS

Defined In: nwdsfilt.h and nwdsfilt.inc

Structure

C

```
typedef struct
{
   pFilter_Node_T   fn ;
   nuint16          level ;
   nuint32          expect ;
} Filter_Cursor_T;
```

Defined In

```
Filter_Cursor_T = Record
      fn : pFilter_Node_T;
      level : nuint16;
      expect : nuint32
End;
```

Fields

fn

Points to the address of the current node structure in the expression tree.

level

Specifies the number of nodes superior to the current node plus 1.

expect

Specifies 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 eDirectory.

Service: NDS

Defined In: nwdsfilt.h and nwdsfilt.inc

Structure

C

```
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

```
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

Points to the address of the parent node. Refers to nodes in relation to the currently selected node.

left

Points to the address of the left subordinate. Refers to nodes in relation to the currently selected node.

right

Points to the address of the right subordinate. Refers to nodes in relation to the currently selected node.

value

Points to the address of an attribute name of attribute value, if token is a value or a name.

syntax

Specifies the syntax associated with the value of token.

token

Specifies the type of node (see Section 5.13, “Filter Tokens,” on page 474).

If token specifies neither an attribute name (14) nor an attribute value (6), value and syntax members are ignored.

Hold_T

Contains the eDirectory information for the attributes that use the Hold syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   pnstr8    objectName ;
   nuint32   amount ;
} Hold_T;
```

Pascal

```
Hold_T = Record
      objectName : pnstr8;
      amount : nuint32
End;
```

Fields

objectName

Points to the distinguished name of the server submitting the hold.

amount

Specifies the amount the server has requested to be held against the user's credit limit.

NDSOSVersion_T

Contains the operating system version information.

Service: NDS

Defined In: nwdsmisc.h

Structure

C

```
typedef struct
{
   nuint32   major;
   nuint32   minor;
   nuint32   revision;
} NDSOSVersion_T, N_FAR *pNDSOSVersion_T;
```

Fields

major

Specifies the major version number.

minor

Specifies the minor version number (the number following the period).

revision

Specifies the revision number.

This structure is used by the DSPING_OS_VERSION flag. For more information, see Section 5.19, “eDirectory Ping Flags,” on page 479.

NDSStatsInfo_T

Contains statistical information for eDirectory relative to an eDirectory server.

Service: NDS

Defined In: nwdsmisc.h and nwdsmisc.inc

Structure

C

```
typedef struct
{
   nuint32   statsVersion ;
   nuint32   noSuchEntry ;
   nuint32   localEntry ;
   nuint32   typeReferral ;
   nuint32   aliasReferral ;
   nuint32   requestCount ;
   nuint32   requestDataSize ;
   nuint32   replyDataSize ;
   nuint32   resetTime ;
   nuint32   transportReferral ;
   nuint32   upReferral ;
   nuint32   downReferral ;
} NDSStatsInfo_T, N_FAR *pNDSStatsInfo_T;
```

Pascal

```
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

Specifies the supported members of the statistics structure as it is expanded.

noSuchEntry

Specifies the number of times name resolution resulted in not locating the entry local to this server.

localEntry

Specifies the number of times name resolution resulted in finding the entry local to this server.

typeReferral

Specifies the number of times name resolution found a local entry, but another replica type was requested.

aliasReferral

Specifies the number of times name resolution responded with an alias referral.

requestCount

Specifies the number of NDS requests received from a remote client (including the eDirectory client agent used for skulking, etc.).

requestDataSize

Specifies the sum of request buffer sizes. This number is likely to wrap (overflow back to a lower number) over time.

replyDataSize

Records the sum of reply buffer sizes. This number is likely to wrap (overflow back to a lower number) over time.

resetTime

Specifies the last time eDirectory statistics were reset. The value consists of a whole number of seconds since 12:00 midnight, January 1, 1970, UTC.

transportReferral

Specifies the number of times name resolution located a local entry, but the referral specified does not have an acceptable transport type.

upReferral

Specifies the number of times name resolution was not walking the tree for the caller, and the referral went "up" the tree.

downReferral

Specifies the number of times name resolution was not walking the tree for the caller, and the referral went "down" the tree.

Net_Address_T

Contains the eDirectory information for the attributes that use the Net Address syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   nuint32   addressType ;
   nuint32   addressLength ;
   pnuint8   address ;
} Net_Address_T;
```

Defined In

```
Net_Address_T = Record
      addressType : nuint32;
      addressLength : nuint32;
      address : pnuint8
End;
```

Fields

addressType

Specifies the type of communications protocol used, such as IPX or IP. See Section 5.21, “Network Address Types,” on page 482.

addressLength

Specifies the address length expressed in bytes. The table lists the addresses with defines.

Define	Value	Transport Type
IPX_ADDRESS_LEN	12	IPX
IP_ADDRESS_LEN	6	IP

address

Points to the hexadecimal address. The address field is stored as a binary string; each 4-bit nibble must be converted to hexadecimal before it can be displayed as a hexadecimal address.

NWDS_TimeStamp_T

Contains the eDirectory information for the attributes that use the Timestamp syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   nuint32   wholeSeconds ;
   nuint32   eventID ;
} NWDS_TimeStamp_T;
```

Pascal

```
NWDS_TimeStamp_T = Record
      wholeSeconds : nuint32;
      eventID : nuint32
End;
```

Fields

wholeSeconds

Specifies the value of the time stamp in whole seconds. Zero equals 12:00 a.m. (midnight), January 1, 1970 GMT.

eventID

Specifies the replica ID and the event ID that further orders events occurring within the same whole-second interval.

See also TimeStamp_T (page 458).

Object_ACL_T

Contains the eDirectory information for the attributes that use the Object ACL syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   pnstr8    protectedAttrName ;
   pnstr8    subjectName ;
   nuint32   privileges ;
} Object_ACL_T;
```

Pascal

```
Object_ACL_T = Record
      protectedAttrName : pnstr8;
      subjectName : pnstr8;
      privileges : nuint32
 End;
```

Fields

protectedAttrName

Points either to the name of the specific attribute to be protected or to one of the following defines:

"[All Attributes Rights]"	DS_ALL_ATTRS_NAME
"[Entry Rights]"	DS_ENTRY_RIGHTS_NAME

subjectName

Points either to the name of the object receiving the rights to the protected object or to one of the following defines:

"[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

Specifies a bit mask identifying specific rights (see Section 5.18, “eDirectory Access Control Rights,” on page 477).

Object_Info_T

Contains information used to maintain objects.

Service: NDS

Defined In: nwdsbuft.h and nwdsbuft.inc

Structure

C

```
typedef struct
{
   nuint32   objectFlags ;
   nuint32   subordinateCount ;
   time_t    modificationTime ;
   char      baseClass [MAX_SCHEMA_NAME_BYTES+2];
} Object_Info_T;
```

Pascal

```
Object_Info_T = Record
   objectFlags : nuint32;
   subordinateCount : nuint32;
   modificationTime : time_t;
   baseClass : Array [0..MAX_SCHEMA_NAME_BYTES+1] Of char;
End;
```

Fields

objectFlags

Specifies the object's entry flags (see Section 5.12, “DSI_ENTRY_FLAGS Values,” on page 473).

subordinateCount

Specifies the number of objects subordinates to the object.

modificationTime

Specifies the time when the object was last modified.

baseClass

Specifies the object class used to create the object.

Octet_List_T

Contains the eDirectory information for the attributes that use the Octet List syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct _octet_list
{
   struct _octet_list N_FAR   *next;
   nuint32                     length;
   pnuint8                     data;
} Octet_List_T;
```

Pascal

```
Octet_List_T = Record
   next: pOctet_List_T;
   length : nuint32;
   data : pnuint8
End;
```

Fields

next

Points to the next string in the list.

length

Specifies the length, in bytes, of the data field.

data

Points to the data string.

Octet_String_T

Contains the eDirectory information for the attributes that use the Octet String syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   nuint32   length ;
   pnuint8   data ;
} Octet_String_T;
```

Pascal

```
Octet_String_T = Record
      length : nuint32;
      data : pnuint8
End;
```

Fields

length

Specifies, in bytes, the length of the string.

data

Points to the string.

Path_T

Contains the eDirectory information for the attributes that use the Path syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   nuint32   nameSpaceType ;
   pnstr8    volumeName ;
   pnstr8    path ;
} Path_T;
```

Pascal

```
Path_T = Record
      nameSpaceType : nuint32;
      volumeName : pnstr8;
      path : pnstr8
End;
```

Fields

nameSpaceType

Specifies the name space of the file name (see Section 5.17, “Name Space Types,” on page 477).

volumeName

Points to the distinguished name of the volume.

path

Points to a file system path, formatted according to the name space.

Replica_Pointer_T

Contains the eDirectory information for the attributes that use the Replica Pointer syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   pnstr8          serverName ;
   nint32          replicaType ;
   nint32          replicaNumber ;
   nuint32         count ;
   Net_Address_T   replicaAddressHint [1];
} Replica_Pointer_T;
```

Pascal

```
Replica_Pointer_T = Record
      serverName : pnstr8;
      replicaType : nint32;
      replicaNumber : nint32;
      count : nuint32;
      replicaAddressHint : Array[0..0] Of Net_Address_T
End;
```

Fields

serverName

Points to the distinguished name of the NetWare server storing the replica.

replicaType

Specifies the capabilities of this copy of the partition (see Section 5.23, “Replica Types,” on page 483).

replicaNumber

Specifies the number of the replica.

count

Specifies the number of Net_Address_T structures.

replicaAddressHint

Specifies the node at which the NetWare server probably exists.

Syntax_Info_T

Contains syntax information.

Service: NDS

Defined In: nwdsbuft.h and nwdsbuft.inc

Structure

C

```
typedef struct
{
   nuint32   ID ;
   nstr8     defStr [MAX_SCHEMA_NAME_BYTES + 2];
   nflag16   flags ;
} Syntax_Info_T;
```

Pascal

```
Syntax_Info_T = Record
    ID : nuint32;
    defStr : Array[1..MAX_SCHEMA_NAME_BYTES+2] Of nint8;
    flags : nflag16
End;
```

Fields

ID

Specifies the numeric representation of the syntax name (see Section 5.26, “Syntax IDs,” on page 487).

defStr

Specifies the byte representation of the syntax name.

flags

Specifies the matching rules for the syntax such as equality, greater than, and less than (see Section 5.25, “Syntax Matching Flags,” on page 486).

TimeStamp_T

Contains the information for the functions that manipulate eDirectory time stamps.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   nuint32   wholeSeconds ;
   nuint16   replicaNum ;
   nuint16   eventID ;
} TimeStamp_T;
```

Pascal

```
TimeStamp_T = Record
      wholeSeconds : nuint32;
      replicaNum : nuint16;
      eventID : nuint16
End;
```

Fields

wholeSeconds

Specifies the whole number of seconds, where zero equals 12:00, midnight, January 1, 1970, UTC.

replicaNum

Specifies the number of the replica on which the event occurred.

eventID

Specifies 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.

This structure is used by the following functions: NWDSExtSyncList, NWDSExtSyncRead, NWDSExtSyncSearch, and NWDSGetAttrValModTime.

Typed_Name_T

Contains the eDirectory information for the attributes that use the Typed Name syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   pnstr8    objectName ;
   nuint32   level ;
   nuint32   interval ;
} Typed_Name_T;
```

Pascal

```
Typed_Name_T = Record
      objectName : pnstr8;
      level : nuint32;
      interval : nuint32
End;
```

Fields

objectName

Points to the distinguished name of the eDirectory object.

level

Specifies the priority of the attribute. This is a relative value assigned by the user.

interval

Specifies the frequency of reference. This is a relative value assigned by the user.

Unknown_Attr_T

Contains the eDirectory information for the attributes that use the Unknown attribute syntax.

Service: NDS

Defined In: nwdsattr.h and nwdsattr.inc

Structure

C

```
typedef struct
{
   pnstr8    attrName ;
   nuint32   syntaxID ;
   nuint32   valueLen ;
   nptr      value ;
} Unknown_Attr_T;
```

Pascal

```
Unknown_Attr_T = Record
      attrName : pnstr8;
      syntaxID : nuint32;
      valueLen : nuint32;
      value : nptr
End;
```

Fields

attrName

Points to the attribute name.

syntaxID

Specifies the syntax used by the attribute (see Section 5.26, “Syntax IDs,” on page 487).

valueLen

Specifies the length of the data contained in the attribute.

value

Points to the attribute’s value (data).

Values

5

This chapter defines the flags, information types, keys, and enumerated data types used by the Novell® eDirectory™ functions.

5.1 Attribute Constraint Flags

Attribute constraint flags give the attribute certain characteristics which restrict the information that can be stored in the data type and which constrain the operations of eDirectory and eDirectory clients.

Flag	C Value	Description
DS_SINGLE_VALUED_ATTR	0x0001	Indicates that the attribute has a single value, with no order implied. If FALSE, the attribute is multi-valued.
DS_SIZED_ATTR	0x0002	Indicates that the attribute has an upper and lower boundary. This can be the length for strings or the value for integers.The first number indicates the lower boundary and the second, the upper boundary. If FALSE, the attribute has no length or range limits.
DS_NONREMOVABLE_ATTR	0x0004	Prevents the attribute from being removed from the schema: ◆ In NDS version 6.xx and below, clients cannot set this constraint flag. ◆ In NDS version 7.xxx and above, clients can set this flag when the attribute is created. All operational attribute definitions have the nonremovable flag set to TRUE. If FALSE, the attribute can be removed if it hasn't been assigned to a class.
DS_READ_ONLY_ATTR	0x0008	Prevents clients from remotely modifying the attribute. The eDirectory server and applications running on it create and maintain these attributes. Clients can read the attribute's value. If FALSE, clients can remotely modify this attribute.
DS_HIDDEN_ATTR	0x0010	In NDS version 6.xx and below, marks the attribute as usable only by the eDirectory server. In NDS version 7.xx and above, marks the attribute as usable by eDirectory and the applications running on the eDirectory server. If FALSE, clients can see the attribute.
DS_STRING_ATTR	0x0020	Labels the attribute as a string type. eDirectory sets this constraint on all attributes that use a string for their syntax. Naming attributes must have this constraint. If FALSE, the attribute is not a string and cannot be used as a naming attribute.

Flag	C Value	Description
DS_SYNC_IMMEDIATE	0x0040	Forces immediate synchronization with other replicas when the value of the attribute changes. If FALSE, the attribute is synchronized at the next synchronization interval. In NetWare 5.x, all attributes in the operational schema have this constraint except Back Link, Bindery Property, Bindery Object Restriction, Bindery Restriction Level, Bindery Type, Last Login Time, Last Referenced Time, Login Time, Purge Vector, Reference, Synchronize Up To, Timezone, Transitive Vector, Unknown, and Unknown Base Class.
DS_PUBLIC_READ	0x0080	Indicates that anyone can read the attribute without read privileges being assigned. You cannot use inheritance masks to prevent an object from reading attributes with this constraint. If FALSE, eDirectory rights determine who can read the value of the attribute. If TRUE, eDirectory skips all rights checking, making access to the data extremely efficient.
DS_SERVER_READ	0x0100	Indicates that Server class objects can read the attribute even though the privilege to read has not been inherited or explicitly granted. You cannot use inheritance masks to restrict servers from reading attributes with this constraint. The client cannot set or modify this constraint flag and thus cannot modify the attribute.
DS_WRITE_MANAGED	0x0200	Forces users to have supervisor rights to the object before they can add or delete the object as a value for this attribute. This flag only works on attributes which have a DN in the syntax. It is used on attributes such as Security Equals, Group Membership, and Profile Membership.
DS_PER_REPLICA	0x0400	Marks the attribute so that the information in the attribute is not synchronized with other replicas. The client cannot set or modify this constraint flag and thus cannot modify the attribute.
DS_SCHEDULE_SYNC_NEVER	0x0800	Allows the attribute’s value to change without such a change triggering synchronization. The attribute can wait to propagate the change until the next regularly scheduled synchronization cycle or some other event triggers synchronization.
DS_OPERATIONAL	0x1000	Indicates that eDirectory uses the attribute internally and requires the attribute to function correctly. Also used for LDAP compatibility.

5.2 Attribute Value Flags

The following flags indicate information about the attribute other than the attribute's value. They are defined in the nwdsdefs.h file.

Name	C Value	Description
DS_NOT_PRESENT	0x0000	Indicates that the attribute does not contain a value.
DS_NAMING	0x0001	Indicates that the attribute is a naming attribute for the object.
DS_BASECLASS	0x0002	Indicates that the attribute is the base class of the object.
DS_PRESENT	0x0004	Indicates that the attribute's value is present.
DS_VALUE_DAMAGED	0x0008	Indicates that the value does not conform to the attribute's defined syntax and is therefore damaged.
DS_SUPERCLASS	0x0010	Indicates the attribute is a super class of the object.
DS_AUXILIARYCLASS	0x0020	Indicates the attribute is an auxiliary class of the object.

5.3 Buffer Operation Types and Related Functions

Functions which use input buffers must allocate the buffer and then initialize it to receive its type of information. The NWDSInitBuf function calls this the operation parameter, because it specifies the type of operation the function using the input buffer performs. These functions are called “Related Functions” in the table below.

The operation type is used both to initialize the buffer and to end an operation if the related function has an iterationHandle parameter. These functions have the potential of returning more information than can fit in an allocated output buffer. To end such an operation before the complete results have been retrieved, call NWDSCloseIteration with the operation type. This function frees the memory and states associated with the operation. The memory and states are automatically freed when all information is retrieved from an operation.

Operation Type	C Value	Related Function
DSV_READ	3	NWDSExtSyncRead (page 150)
		NWDSListAttrsEffectiveRights (page 251)
		NWDSRead (page 327)
DSV_COMPARE	4	NWDSCompare (page 128)
DSV_LIST	5	NWDSList (page 248)
DSV_SEARCH	6	NWDSExtSyncList (page 146)
		NWDSExtSyncSearch (page 154)
		NWDSListByClassAndName (page 254)
		NWDSListContainers (page 261)
		NWDSSearch (page 383)
DSV_ADD_ENTRY	7	NWDSAddObject (page 87)

Operation Type	C Value	Related Function
DSV_MODIFY_ENTRY	9	NWDSModifyObject (page 286)
DSV_READ_ATTR_DEF	12	NWDSReadAttrDef (page 330)
DSV_DEFINE_CLASS	14	NWDSDefineClass (page 137)
DSV_READ_CLASS_DEF	15	NWDSReadClassDef (page 333)
DSV_MODIFY_CLASS_DEF	16	NWDSModifyClassDef (page 281)
DSV_LIST_CONTAINABLE_CLASSES	18	NWDSListContainableClasses (page 258)
DSV_LIST_PARTITIONS	22	NWDSListPartitions (page 264)
		NWDSListPartitionsExtInfo (page 267)
DSV_SEARCH_FILTER	28	NWDSPutFilter (page 323)
DSV_READ_SYNTAXES	40	NWDSReadSyntaxes (page 348)
DSV_BACKUP_ENTRY	45	NWDSBackupObject (page 107)
DSV_RESTORE_ENTRY	46	NWDSRestoreObject (page 373)
DSV_READ_REFERENCES	79	NWDSReadReferences (page 342)

5.4 Class Flags

Besides basic information about containment classes, naming attributes, mandatory and optional attributes, and super classes, eDirectory maintains a set of flags that further define the class object.

The following table lists this set of class flags.

Flag	C Value	Comment
DS_CONTAINER_CLASS	0x01	If TRUE, objects of the class can have subordinates.
DS_EFFECTIVE_CLASS	0x02	If TRUE, the class can be used as a base class. If FALSE, the class is a noneffective class which can be used as a super class of effective classes.
DS_NONREMOVABLE_CLASS	0x04	If TRUE, the class definition can't be removed from the schema. All classes defined by the base schema have this flag set to TRUE. For classes that extend the schema, eDirectory sets this flag to TRUE when an entry is created from the class.
DS_AMBIGUOUS_NAMING	0x08	If TRUE, the class can't be used as a base class. Noneffective classes can be created with ambiguous naming. eDirectory sets this flag when the class is created.

Flag	C Value	Comment
DS_AMBIGUOUS_CONTAINMENT	0x10	If TRUE, the class can't be used as a base class. Noneffective classes can be created with ambiguous naming eDirectory sets this flag when the class is created.
DS_AUXILIARY_CLASS	0x20	If TRUE, the class is an auxiliary class. The DS_CONTAINER_CLASS and the DS_EFFECTIVE_CLASS flags must be set to FALSE. This is a new flag for NDS 8.
DS_OPERATIONAL_CLASS	0x40	Defined for internal eDirectory use in NDS 8 and for LDAP compatibility.

For a more detailed description of these flags, see "Object Class Flags" (*NDK: Novell eDirectory Schema Reference*).

5.5 Change Types for Modifying Objects

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. These change types are used with the NWDSPutChange and NWDSPutChangeAndVal functions. The change type flags have subtle differences. Select the flag according to the error conditions you want reported.

Change Type	C Value	Description
DS_ADD_ATTRIBUTE	0x00	Adds the first instance of an attribute to an object. Adding an attribute requires the attribute name and value. An attempt to add an already existing attribute results in an error. A modify operation using this flag should be preceded by a read operation to ensure that the attribute does not already exists.
DS_REMOVE_ATTRIBUTE	0x01	Removes an attribute from an object. The following conditions return errors: ◆ The attribute is not present. ◆ The attribute is present in the RDN.
DS_ADD_VALUE	0x02	Adds a value to an attribute. Adding values requires the attribute name and value. Attribute values inserted in the buffer following the change record are added to the specified attribute. An attempt to add a value to a nonexistent attribute succeeds. An attempt to add an already existing value results in an error.

Change Type	C Value	Description
DS_REMOVE_VALUE	0x03	Removes values from an attribute. Removing values requires the attribute name and value. Attribute values put in the buffer following this change record are removed from the specified attribute. The following conditions return errors: ◆ The value is not present in the attribute ◆ The value is present in the RDN.
DS_ADDITIONAL_VALUE	0x04	Adds an additional new value to a multivalued attribute. The following conditions return errors: ◆ The attribute does not already have a value ◆ The value matches an exisiting value
DS_OVERWRITE_VALUE	0x05	Modifies an attribute value without needing to remove the old value first and then add the new value. ◆ If the attribute is single-valued, deletes the old value, and adds the new value. If the old and new value are the same, eDirectory updates the value's timestamp but does not return an error. ◆ If the attribute is multivalued, adds the new value, leaving the old values. If the new value already exists, eDirectory updates the value's timestamp but does not return an error. This flag can be use to add the first value to an attribute (single valued or multivalued).
DS_CLEAR_ATTRIBUTE	0x06	Deletes an attribute. If the attribute doesn't exist, does not report an error.
DS_CLEAR_VALUE	0x07	Clears an attribute value. If the value does not exists, does not report an error.

5.6 Context Keys and Flags

eDirectory context keys are used when setting and getting context handle information. They are defined in the nwdsdc.h file. Most keys can be set with bit mask flags which modify the information returned by a key. For an overview of how the keys and flags work together, see Section 1.1, “Context Handles,” on page 15.

The functions for manipulating information about the NDS context are NWDSGetContext (page 191) and NWDSSetContext (page 387). Both require a key value as a parameter, and the table below lists the defines available for the key parameter. For the default values, see Section 5.7, “Default Context Key Values,” on page 469.

Key Name and C Value	Data Type	Description
DCK_FLAGS 1	nuint32	Determines how requests to eDirectory are processed and how data is returned (see Section 5.8, “DCK_FLAGS Bit Values,” on page 470).
DCK_CONFIDENCE 2	nuint32	Determines replica type when processing requests (see Section 5.10, “DCK_CONFIDENCE Bit Values,” on page 471).
DCK_NAME_CONTEXT 3	NULL terminated string	Contains the current location in the eDirectory tree. For the format of the string, see the DCV_XLATE_STRINGS flag in Section 5.8, “DCK_FLAGS Bit Values,” on page 470.
DCK_TRANSPORT_TYPE 4		Not currently used.
DCK_REFERRAL_SCOPE 5		Not currently used.
DCK_LAST_CONNECTION 8	NWCONN_HANDLE	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_LAST_SERVER_ADDRESS 9		NLM only. Obsolete. Use DCK_LAST_CONNECTION instead.
DCK_LAST_ADDRESS_USED 10		NLM only. Obsolete. Use DCK_LAST_CONNECTION instead.
DCK_TREE_NAME 11	NULL terminated string	Contains the name of the tree in the current context. This name must be the literal eDirectory tree name. DNS names or string forms of a network address are not supported. For the format of the string, see the DCV_XLATE_STRINGS flag in Section 5.8, “DCK_FLAGS Bit Values,” on page 470.
DCK_DSI_FLAGS 12	nuint32	Determines the eDirectory object information to be returned by the NWDSList (page 248), NWDSReadObjectDSIInfo (page 338), NWDSReadObjectInfo (page 340), and NWDSSearch (page 383) functions (see Section 5.11, “DCK_DSI_FLAGS Values,” on page 472).

Key Name and C Value	Data Type	Description
DCK_NAME_FORM 13	nuint32	Determines whether eDirectory returns distinguished names in partial dot or slash format (see Section 5.9, “DCK_NAME_FORM Values,” on page 471).
DCK_NAME_CACHE_DEPTH 15	nuint32	Determines how many eDirectory names are kept in the cache. When the cache is full, the oldest cache record is dropped.

Before changing the DCK_FLAGS information, first read the current flags by calling NWDSGetContext (page 191). Use bitwise operations to change the flag(s) you want to change while leaving the settings of the other flags unchanged. Then call the NWDSSetContext function to set DCK_FLAGS to the desired settings.

Related Topics:

- “Modifying the Context of the Context Handle” on page 50
- “Reading the Context of the Context Handle” on page 51

5.7 Default Context Key Values

When a context handle is created, it is initialized to the following default values. To modify the default values, use the NWDSGetContext (page 191) and the NWDSSetContext (page 387) functions. To understand the default values, see Section 5.6, “Context Keys and Flags,” on page 467.

Key Name	C Value	Default Value
DCK_FLAGS	1	DCV_DEREF_ALIASES | DCV_XLATE_STRINGS | DCV_CANONICALIZE_NAMES
DCK_CONFIDENCE	2	DCV_LOW_CONF
DCK_NAME_CONTEXT	3	For client applications, the default value for the client machine. For the NLM platform, the bindery context.
DCK_LAST_CONNECTION	8	For client applications, the connection to the server in the eDirectory tree that was used previously to service eDirectory operations. For the NLM platform, initialized to no connection.
DCK_TREE_NAME	11	For client applications, the preferred tree of the client machine. For the NLM platform, the tree of the local server.
DCK_DSI_FLAGS	12	DSI_ENTRY_FLAGS | DSI_OUTPUT_FIELDS | DSI_SUBORDINATE_COUNT | DSI_MODIFICATION_TIME | DSI_BASE_CLASS | DSI_ENTRY_RDN | DSI_ENTRY_DN

Key Name	C Value	Default Value
DCK_NAME_FORM	13	DCV_NF_PARTIAL_DOT
DCK_NAME_CACHE_DEPTH	15	5

5.8 DCK_FLAGS Bit Values

The following table lists the bit values that the DCK_FLAGS key uses. They can be ORed together to specify the kinds of information the NWDSGetContext (page 191) or NWDSSetContext (page 387) functions set or retrieve. They are defined in the nwdsdc.h file.

Flag Name	C Value	Description
DCV_DEREF_ALIASES	0x00000001L	When set to one (1), deferences alias objects. This means that information about the object referenced by the alias, rather than alias object, is returned. When set to zero (0), returns information about the Alias, rather than the object it references. Default value: 1
DCV_XLATE_STRINGS	0x00000002L	When set to one (1), translates from Unicode strings to the local code page. When set to zero (0), returns Unicode strings. Array element size is byte for ASCII characters or double-byte for Unicode characters. Default value: 1
DCV_TYPELESS_NAMES	0x00000004L	When set to one (1), returns typeless names. When set to zero (0), returns typeful names. Default value: 0
DCV_ASYNC_MODE	0x00000008L	Reserved.
DCV_CANONICALIZE_NAMES	0x00000010L	When set to one (1), the object name is appended to the name context value to form a distinguished name before sending a request to eDirectory. Partial names, relative to the name context, are returned. When set to zero (0), no name manipulation is performed. Applications must send eDirectory distinguished names. Default value: 1

Flag Name	C Value	Description
DCV_DEREF_BASE_CLASS	0x00000040L	When set to one (1), returns the base class value of the object the alias references. When set to zero (0), returns Alias as the base class of Alias objects. This flag affects List, Read, and Search operations. Default value: 0
DCV_DISALLOW_REFERRALS	0x00000080L	When set to one (1), referrals sent by eDirectory are not used and the current eDirectory agent must resolve the name. Default value: 0
DCV_ALWAYS_EVALUATE_REFERRALS	0x00000100L	When set to one (1), referrals are always requested and evaluated evein if the current eDirectory agent has a replica of the requested object. Applications can set this value to always attempt to communicate with the closest server. Default value: 0

5.9 DCK_NAME_FORM Values

The DCK_NAME_FORM values determine the format of the eDirectory names. They are defined in the nwdsdc.h file.

Flag Name	C Value	Description
DCV_NF_PARTIAL_DOT	1	eDirectory returns names in partial dot form. This is the default value.
DCV_NF_SLASH	3	eDirectory returns names in slash form.

For more information and examples of the format, see "DCK_NAME_FORM Key" on page 22.

5.10 DCK_CONFIDENCE Bit Values

The following table lists the bit values that the DCK_CONFIDENCE key uses. Select one to specify the kinds of information the NWDSGetContext (page 191) or NWDSSetContext (page 387) functions set or retrieve. They are defined in the nwdsdc.h file.

Flag Name	C Value	Description
DCV_LOW_CONF	0	Allows information to be obtained from any replica. This is the default value.
DCV_MED_CONF	1	Allows information to be obtained from any replica (same as DCV_LOW_CONF).
DCV_HIGH_CONF	2	Requires information to be obtained from the master replica.

5.11 DCK_DSI_FLAGS Values

The following table lists the flags the DCK_DSI_FLAGS key uses. They are defined in the nwdsdc.h file. These DSI flags affect the information returned by the NWDSReadObjectInfo (page 340), NWDSList (page 248), NWDSReadObjectDSIInfo (page 338), and NWDSGetDSIInfo (page 199) functions.

If you remove the default flags from the context handle, the NWDSReadObjectInfo (page 340) and NWDSGetDSIInfo (page 199) functions that follow the NWDSList (page 248) function will be affected.

If you add flags to the context handle, you can access the additional information by calling the NWDSReadObjectDSIInfo (page 338) function or by calling the NWDSList (page 248), NWDSGetObjectNameAndInfo (page 217), and NWDSGetDSIInfo (page 199) functions in this specific order.

Flag Name and C Value	Data Type	Description
DSI_OUTPUT_FIELDS 0x00000001L	nuint32	Specifies which requested fields of information are actually returned. If the server cannot provide one of the items requested by the DSI flags, the server clears the corresponding bit in the Output Fields flag. No matter which bits are set, the order of the returned data is the same as the requested order.
DSI_ENTRY_ID 0x00000002L	nuint32	Returns the Entry ID of the object.
DSI_ENTRY_FLAGS: 0x00000004L	nuint32	Returns the type of entry. See Section 5.12, “DSI_ENTRY_FLAGS Values,” on page 473.
DSI_SUBORDINATE_COUNT 0x00000008L	nuint32	Returns the number of objects that are subordinate to the specified object, or 0 if unknown.
DSI_MODIFICATION_TIME 0x00000010L	nuint32	Returns when the object was lasted modified, or 0 if unknown.
DSI_MODIFICATION_TIMESTAMP 0x00000020L	Timestamp_T	Returns the object’s modification timestamp, or 0 if unknown.
DSI_CREATION_TIMESTAMP 0x00000040L	Timestamp_T	Returns the object’s creation timestamp, or 0 if unknown.
DSI_PARTITION_ROOT_ID 0x00000080L	nuint32	Returns the Entry ID of the partition’s root entry.
DSI_PARENT_ID 0x00000100L	nuint32	Returns the Entry ID of the object’s parent.
DSI_REVISION_COUNT 0x00000200L	nuint32	Returns the number of times the object has been modified.

Flag Name and C Value	Data Type	Description
DSI_REPLICA_TYPE 0x00000400L	nuint32	Returns the replica type: ◆ 0 RT_MASTER ◆ 1 RT_SECONDARY ◆ 2 RT_READONLY ◆ 3 RT_SUBREF
DSI_BASE_CLASS 0x00000800L	NULL terminated string	Returns the base class of the object.
DSI_ENTRY_RDN 0x00001000L	NULL terminated string	Returns the relative distinguished name, or partial name, of the object in the format specified by DCK_FLAGS and DCK_NAME_FORM (see Section 5.6, “Context Keys and Flags,” on page 467).
DSI_ENTRY_DN 0x00002000L	NULL terminated string	Returns the distinguished name of the object in the format specified by the DCK_FLAGS (see Section 5.6, “Context Keys and Flags,” on page 467). The name form is always comma delimited (not slash delmited), regardless of the setting for DCK_NAME_FORM.
DSI_PARTITION_ROOT_DN 0x00004000L	NULL terminated string	Returns the distinguished name of the partition root in the format specified by the DCK_FLAGS and DCK_NAME_FORM (see Section 5.6, “Context Keys and Flags,” on page 467).
DSI_PARENT_DN 0x00008000L	NULL terminated string	Returns the distinguished name of the object’s parent in the format specified by the DCK_FLAGS and DCK_NAME_FORM (see Section 5.6, “Context Keys and Flags,” on page 467).
DSI_PURGE_TIME 0x00010000L	nuint32	For a partition object, returns the oldest purge time.
DSI_DEREFERENCE_BASE_CLASS 0x00020000L	NULL terminated string	Returns the base class of the object the alias references.
DSI_REPLICA_NUMBER 0x00040000L	nuint32	Returns the number the replica was assigned when it was created.
DSI_REPLICA_STATE 0x00080000L	nuint32	Returns the state of the replica (see Section 5.24, “Replica States,” on page 484).

5.12 DSI_ENTRY_FLAGS Values

DSI_ENTRY_FLAGS returns information about an entry’s state. The flags are defined in the nwdsdefs.h file. The following table lists the flags that have been defined for the various entry states.

Flag Name	C Value	Description
DS_ALIAS_ENTRY	0x0001	Indicates that the entry is an alias object.
DS_PARTITION_ROOT	0x0002	Indicates that the entry is the root partition.
DS_CONTAINER_ENTRY	0x0004	Indicates that the entry is a container object and not a container alias.
DS_CONTAINER_ALIAS	0x0008	Indicates that the entry is a container alias.
DS_MATCHES_LIST_FILTER	0x0010	Indicates that the entry matches the List filter.
DS_REFERENCE_ENTRY	0x0020	Indicates that the entry has been created as a reference rather than an entry. The synchronization process is still running and has not created an entry for the object on this replica.
DS_40X_REFERENCE_ENTRY	0x0040	Indicates that the entry is a reference rather than the object. The reference is in the older 4.0x form and appears only when upgrading.
DS_BACKLINKED	0x0080	Indicates that the entry is being back linked.
DS_NEW_ENTRY	0x0100	Indicates that the entry is new and replicas are still being updated.
DS_TEMPORARY_REFERENCE	0x0200	Indicates that an external reference has been temporarily created for authentication; when the object logs out, the temporary reference is deleted.
DS_AUDITED	0x0400	Indicates that the entry is being audited.
DS_ENTRY_NOT_PRESENT	0x0800	Indicates that the state of the entry is not present.
DS_ENTRY_VERIFY_CTS	0x1000	Indicates the entry's creation timestamp needs to be verified. eDirectory sets this flag when a replica is removed or upgraded from NetWare 4.11 to NetWare 5.
DS_ENTRY_DAMAGED	0x2000	Indicates that the entry's information does not conform to the standard format and is therefore damaged.

5.13 Filter Tokens

Filter tokens are used in search expression trees for NWDSSearch and NWDSItrCreateSearch. They are defined in the nwdsfilt.h file. For more information on creating a search expression tree, see Section 1.4, “Search Requests,” on page 30.

Token	C Value	Description
FTOK_END	0	Signals the end of the search expression tree.
FTOK_OR	1	Indicates TRUE if either of the subordinate nodes are true.

Token	C Value	Description
FTOK_AND	2	Indicates TRUE only if both subordinate nodes are true
FTOK_NOT	3	Indicates TRUE if the node is false
FTOK_LPAREN	4	Indicates a left parenthesis for nesting of conditions.
FTOK_RPAREN	5	Indicates a right parenthesis for nesting of conditions
FTOK_AVAL	6	Indicates a value follows.
FTOK_EQ	7	Indicates TRUE only if the attribute's value is equal to the asserted value.
FTOK_GE	8	Indicates TRUE only if the attribute's value is greater than or equal to the asserted value.
FTOK_LE	9	Indicates TRUE only if all attribute's values are less than the asserted value.
FTOK_APPROX	10	Indicates TRUE only if the value of the attribute matches the asserted value. If the attribute syntax does not support approximate matching, this token matches for equality.
FTOK_ANAME	14	Indicates a name follows.
FTOK_PRESENT	15	Indicates TRUE only if the attribute is present on the entry.
FTOK_RDN	16	Indicates TRUE only if the entry's RDN matches the asserted value.
FTOK_BASECLS	17	Indicates TRUE only if the entry belongs to the asserted base class.
FTOK_MODTIME	18	Indicates TRUE only if the modification time stamp is greater than or equal to the asserted value.
FTOK_VALTIME	19	Indicates TRUE only if the creation time stamp is greater than or equal to the asserted value.

5.14 Information Types for Attribute Definitions

The NWDSReadAttrDef (page 330) function uses the following information types to determine whether information is returned about the attribute names or also about attribute characteristics.

Name	C Value	Description
DS_ATTR_DEF_NAMES	0	Returns only the name of the attributes
DS_ATTR_DEFS	1	Returns the name and the characteristics of the attributes (constraints, syntax ID, value limits, ASN.1 ID)

5.15 Information Types for Class Definitions

The NWDSReadClassDef (page 333) function uses the following information types to determine whether to return information about both class name and class definitions. They also determine whether to return information about expanded class-definitions, ASN.1 IDs and default ACLs.

Name	C Value	Description
DS_CLASS_DEF_NAMES	0	Returns only the class names.
DS_CLASS_DEFS	1	Returns class names, class flags, and class definitions (super classes, containment classes, naming attributes, mandatory attributes, and optional attributes).
DS_EXPANDED_CLASS_DEFS	2	Returns class names, class flags, class definitions, and class definitions of the super classes.
DS_INFO_CLASS_DEFS	3	Returns class names, class flags, and ASN.1 identifiers.
DS_FULL_CLASS_DEFS	4	Returns class names, class flags, class definitions, class definitions of the super classes, and default ACLs.

5.16 Information Types for Search and Read

The following flags are found in the nwdsdefs.h file and specify the type of information to return. The NWDSSearch, NWDSReadReferences, NWDSExtSyncRead, NWDSExtSyncSearch, and NWDSItrCreateSearch functions support only the first two types: DS_ATTRIBUTE_NAMES and DS_ATTRIBUTE_VALUES. The NWDSRead function supports all of them.

All the functions that use these information types return the information to an output buffer. The information must be retrieved from the buffer with specialized functions. All of the information types require the use of two functions: NWDSGetAttrCount and NWDSGetAttrName. Use the NWDSGetAttrCount to determine the number of attributes associated with the object. Use the NWDSGetAttrName function to retrieve the name and the number of values associated with the attribute. If only names have been requested, the number of values will always be zero. If the information type returns additional information, you will need to use the specialized functions to retrieve the additional information before retrieving the next attribute name.

Name	C Value	Description
DS_ATTRIBUTE_NAMES	0x00	Returns only the names of the attributes.
DS_ATTRIBUTE_VALUES	0x01	Returns the names and the values of the attributes. Use the NWDSGetAttrVal function to retrieve the values. Call this function once for each value associated with the attribute before getting the name of the next attribute. If the value's size is unknown (most attributes have known sizes), use NWDSComputeAttrValSize before retrieving the value.

Name	C Value	Description
DS_EFFECTIVE_PRIVILEGES	0x02	Returns the names and effective privileges of the requester to the attributes.
DS_VALUE_INFO	0x03	Returns the attribute name, number of values, value flags (see Section 5.2, “Attribute Value Flags,” on page 463), timestamp, and value. Use the NWDSGetAttrValFlags function to retrieve the attribute value flag information. Use the NWDSGetAttrValModTime function to retrieve the modification timestamp for the value.
DS_ABBREVIATED_VALUE	0x04	Returns the attribute name, number of values, attribute value flags (see Section 5.2, “Attribute Value Flags,” on page 463), timestamp, and length of the value. Use the NWDSGetAttrValFlags function to retrieve the attribute value flag information. Use the NWDSGetAttrValModTime function to retrieve the modification timestamp.
DS_EXPANDED_CLASS	0x08	Reserved.

5.17 Name Space Types

The name space types used by the Path_T (page 455) structure are listed in the NAME_SPACE_TYPE typedef enumeration in the nwdsdefs.h and nwdsdefs.inc files. These values match the standard name space types defined for the name space functions (see Name Space Flag Values (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/hvxfva0i.html) (Multiple and Inter-File Services (http://developer.novell.com/ndk/doc/clib/index.html?page=/ndk/doc/clib/cmgnxenu/data/hvxfva0i.html))).

The following table lists this set of name space flags.

Flag	C Value	Comment
DS_DOS	0	DOS name space
DS_MACINTOSH	1	Macintosh name space
DS_UNIX	2	UNIX or NFS name space
DS_FTAM	3	FTAM name space
DS_OS2	4	OS/2, Windows 95, or Windows NT name space

5.18 eDirectory Access Control Rights

eDirectory uses two types of rights: object rights and attribute rights. The object that receives the rights is called a trustee. Except for the Inheritance Control rights (DS_ENTRY_INHERIT_CTL and DS_ATTR_INHERIT_CTL), set the right in the bit mask to 1 (one) to grant the right and to 0 (zero) to deny the right. For the Inheritance Control rights, see Table 5-3 on page 479 for the correct settings.

eDirectory uses the following rights in an ACL to grant rights to the object as a whole. These rights are ORed together into a bit mask.

Table 5-1 *Object Rights*

Flag Name	C Value	Description
DS_ENTRY_BROWSE	0x00000001L	Allows a trustee to discover objects in the eDirectory tree.
DS_ENTRY_ADD	0x00000002L	Allows a trustee to create child objects (new objects that are subordinate to the object in the tree).
DS_ENTRY_DELETE	0x00000004L	Allows a trustee to delete an object. This right does not allow a trustee to delete a container object that has subordinate objects.
DS_ENTRY_RENAME	0x00000008L	Allows a trustee to rename the object.
DS_ENTRY_SUPERVISOR	0x00000010L	Gives a trustee all rights to an object and its attributes.
DS_ENTRY_INHERIT_CTL	0x00000040L	Allows a trustee to inherit the rights granted in the ACL and exercise them on subordinate objects. For information on setting the bit values, see Table 5-3 on page 479.

eDirectory uses the following rights in an ACL to grant rights to individual attributes and to [All Attributes Rights] of an object. These rights are ORed together into a bit mask.

Table 5-2 *Attribute Rights*

Flag Name	C Value	Description
DS_ATTR_COMPARE	0x00000001L	Allows a trustee to compare a value with an attribute's value. This allows the trustee to see if the attribute contains the value without having rights to see the value.
DS_ATTR_READ	0x00000002L	Allows a trustee to read an attribute value. This right confers the Compare right.
DS_ATTR_WRITE	0x00000004L	Allows a trustee to add, delete, or modify an attribute value. This right also gives the trustee the Self (Add or Delete Self) right.
DS_ATTR_SELF	0x00000008L	Allows a trustee to add or delete its name as an attribute value on those attributes that take object names as their values.
DS_ATTR_SUPERVISOR	0x00000020L	Gives a trustee all rights to the object's attributes.
DS_ATTR_INHERIT_CTL	0x00000040L	Allows a trustee to inherit the rights granted in the ACL and exercise these attribute rights on subordinate objects. For information on setting the bit values, see Table 5-3 on page 479.

The bit settings for the Inheritance Control rights use values that ensure compatibility with NetWare 4.x.

Table 5-3 *Inheritance Control Settings*

NetWare Version	Object Right DS_ENTRY_INHERIT_CTL	[All Attributes Rights] DS_ATTR_INHERIT_CTL	Specific Attribute DS_ATTR_INHERIT_CTL
NetWare 4.x	NetWare 4.x does not support this functionality. Inheritance of object rights is always supported. NetWare 4.x requires this bit to be set to 0.	NetWare 4.x does not support this functionality. Inheritance of rights to [All Attributes Rights] is always supported. NetWare 4.x requires this bit to be set to 0.	NetWare 4.x does not support this functionality. Inheritance of ACLs to specific attributes is always blocked. NetWare 4.x requires this bit to be set to 0.
NetWare 5.x	NetWare 5.x supports this right. Set this bit to 0 (zero) to allow the inheritance of the rights in the ACL. Set this bit to 1 (one) to block the inheritance of the ACL rights.	NetWare 5.x supports this right. Set this bit to 0 (zero) to allow the inheritance of the rights granted to [All Attributes Rights]. Set this bit to 1 (one) to block the inheritance of the ACL rights.	NetWare 5.x supports this right. Set this bit to 1 (one) to allow the inheritance of the rights granted to the specific attribute. Set this bit to 0 to block the inheritance of the ACL rights.

5.19 eDirectory Ping Flags

Since a server may not be able to reply to all the requested DSPING flags, the DSPING_SUPPORTED_FIELDS should always be one of the flags ORed together when using the NWDSReadNDSInfo (page 336) function.

Table 5-4 *eDirectory Ping Input Values*

Flag Name and C Value	Data Type	Meaning
DSPING_SUPPORTED_FIELDS 0x00000001L	nuint32	Specifies which requested fields of information are actually returned. If the server cannot provide one of the items requested by the DSPING flags, the server clears the corresponding bit in the Supported Fields flag. No matter which bits are set, the order of the returned data is the same as the requested order.
DSPING_DEPTH 0x00000002L	nuint32	Returns the number of levels down from the eDirectory tree's root object to the server's root-most entry.
DSPING_BUILD_NUMBER 0x00000004L	nuint32	Returns the internal NDS/eDirectory version number.

Flag Name and C Value	Data Type	Meaning
DSPING_FLAGS 0x00000008L	nuint32	See Table 5-5 on page 480.
DSPING_VERIFICATION_FLAGS 0x00000010L	nuint32	See Table 5-6 on page 481.
DSPING_LETTER_VERSION 0x00000020L	nuint32	Returns the internal eDirectory letter that may accompany the NDS/eDirectory version number.
DSPING_OS_VERSION 0x00000040L	NDSOSVersion_T (page 446) structure	Returns three nuint32 values; the first is the major version number of the operating system; the second, the minor version number; the third, the revision number.
DSPING_TIMESYNC_STATE 0x00000080L		Not used.
DSPING_LICENSE_FLAGS 0x00000100L	nuint32	Not used; always returns zero (0).
DSPING_DS_TIME 0x00000200L	nuint32	Returns the current UTC time of the server (number of seconds since January 1, 1970).
DSPING_SAP_NAME 0x00010000L	NULL terminated string	Returns the name the eDirectory server is using for SAP protocol.
DSPING_TREE_NAME 0x00020000L	NULL terminated string	Returns the name of the eDirectory tree the server belongs to.
DSPING_OS_NAME 0x00040000L	NULL terminated string	Returns the name of the operating system which eDirectory is running on top of.
DSPING_HARDWARE_NAME 0x00080000L	NULL terminated string	Returns the hardware of the server's machine (for example, PC compatible).
DSPING_VENDOR_NAME 0x00100000L	NULL terminated string	Returns the name of the company that produced this version of the eDirectory server. Usually, Novell, Inc.

The DSPING_FLAGS can be ORed together.

Table 5-5 *DSPING_FLAGS Values*

Flag Name	C Value	Meaning
DSPONG_ROOT_MOST_MASTER	0x0001	If set to one (1), the server contains the master replica of the root-most entry.
DSPONG_TIME_SYNCHRONIZED	0x0002	If set to one (1), the time on the server is synchronized.

The DSPING_VERIFICATION_FLAGS can be ORed together.

Table 5-6 *DSPING_VERIFICATION_FLAGS Values*

Flag Name	C Value	Meaning
DSPING_VERIFICATION_CHECKSUM	0x00000001 L	If set to one (1), the server is doing IPX checksums.
DSPING_VERIFICATION_CRC32	0x00000002 L	If set to one (1), the server is doing CRC error checking.

5.20 DSP Replica Information Flags

The following table lists the DSP flags. These flags affect the information returned by NWDSListPartitionsExtInfo (page 267), NWDSGetPartitionExtInfoPtr (page 222), and NWDSGetPartitionExtInfo (page 220).

You can access the additional information by calling NWDSListPartitionsExtInfo (page 267), NWDSGetPartitionExtInfoPtr (page 222), and NWDSGetPartitionExtInfo (page 220) in this specific order.

If the client does not request the DSP_OUTPUT_FIELDS flag, the server must return exactly the information requested in the DSP information flags. If the server does not support the information of a requested flag, an error results. If the client requests the DSP_OUTPUT_FIELDS flag, the server can skip the flags it doesn't support, and the server informs the client of the skip by clearing the corresponding bit in the DSP_OUTPUT_FIELDS flag.

Flag Name and C Value	Data Type	Meaning
DSP_OUTPUT_FIELDS 0x00000001L	nuint32	Specifies which requested fields of information are actually returned. If the server cannot provide one of the items requested by the DSP flags, the server clears the corresponding bit in the DSP_OUTPUT_FIELDS flag. The client tests the bits in the DSP_OUTPUT_FIELDS flag to determine which fields are present in the reply. No matter which bits are set, the order of the returned data is the same as the requested order.
DSP_PARTITION_ID 0x00000002L	nuint32	Returns the Entry ID of the partition's root object on the local server.
DSP_REPLICA_STATE 0x00000004L	nuint32	Returns the current state of the replica (see Section 5.24, "Replica States," on page 484).
DSP_MODIFICATION_TIMESTAMP 0x00000008L	NWDS_TimeStamp_T (page 450) structure	Returns the time and the event ID for the most recent modification to the replica.

Flag Name and C Value	Data Type	Meaning
DSP_PURGE_TIME 0x00000010L	nuint32	Returns the time at which all data has been synchronized. Data, scheduled for deletion, that predates this time, can now be deleted.
DSP_LOCAL_PARTITION_ID 0x00000020L	nuint32	Returns the Entry ID of the partition.
DSP_PARTITION_DN 0x00000040L	NULL terminated string	Returns the distinguished name of the partition. Buffer size should be MAX_DN_BYTES.
DSP_REPLICA_TYPE 0x00000080L	nuint32	Returns the replica's state (see Section 5.24, "Replica States," on page 484) in the high 16 bits and the replica's type (see Section 5.23, "Replica Types," on page 483) in the low 16 bits.
DSP_PARTITION_BUSY 0x00000100L	nbool	Indicates whether the partition is busy merging a tree or moving a subtree. When set to one (1), the partition is busy. When set to zero (0), the partition is not busy.

5.21 Network Address Types

The network address types are enumerated data types which are defined in the nwdsdefs.h and nwdsdefs.inc files.

Name	C Value	Description
NT_IPX	0	Internet Packet Exchange (IPX) network address
NT_IP	1	Internet Protocol (IP) network address
NT_SDLC	2	Synchronous Data Link Control (SDLC) address
NT_TOKENRING_ETHERNET	3	Token ring on Ethernet MAC address
NT_OSI	4	Open Systems Interconnection (OSI) address
NT_APPLETALK	5	AppleTalk network address
NT_NETBEUI	6	NetBIOS Extended User Interface (NetBEUI) address
NT_SOCKADDR	7	Socket address
NT_UDP	8	User Datagram Protocol (UDP) address
NT_TCP	9	Transmission Control Protocol (TCP) address
NT_UDP6	10	User Datagram Protocol (UDP), version 6, address

Name	C Value	Description
NT_TCP6	11	Transmission Control Protocol (TCP), version 6, address
NT_INTERNAL	12	Reserved
NT_URL	13	Uniform Resource Locator address (added for NDS 8)
NT_COUNT	14	Returns the maximum number of network address types defined.

The NT_URL address type was added so that LDAP referrals could be integrated with eDirectory and added to eDirectory referral lists. The address conforms to the format of all other network address types: uint32 for type, uint32 for the size in bytes of the address, and a data buffer for the address. Since a standard URL is a string with UTF-8 characters, the LDAP server in NDS 8 converts the URL address to Unicode and stores it in eDirectory as a Unicode string. See RFC 2044, RFC 2255, RFC 1738 for more information.

5.22 Scope Flags

The NWDSSearch, NWDSExtSynSearch, and NWDSItrCreateSearch functions use these flags.

Flag Name	C Value	Description
DS_SEARCH_ENTRY	0	Search applies only to the base object.
DS_SEARCH_SUBORDINATES	1	Search applies only to the immediate subordinates of the base object.
DS_SEARCH_SUBTREE	2	Search applies to the base object and all of its subordinates. For the NWDSItrCreateSearch function, the search of subordinates is restricted to the subordinates that exist on the server's replicas.
DS_SEARCH_PARTITION	3	Used only by NDS 8 and later versions. Search applies to the base object and all of its subordinates in the partition. If a subordinate partition exists, the search does not continue to those objects.

5.23 Replica Types

The replica types identify the type of replica and are defined in the REPLICA_TYPE typedef enumeration in the nwdsdefs.h file. Replica type determines the types of client operations that can be performed on the replica.

Flag Name	C Value	Meaning
RT_MASTER	0	Identifies this replica as the master replica of the partition. Entries can be modified; partition operations can be performed.

Flag Name	C Value	Meaning
RT_SECONDARY	1	Identifies this replica as a secondary replica of the partition. Secondary replicas are Read/Write replicas and entries can be modified.
RT_READONLY	2	Identifies the replica as a Read-Only replica. Only the eDirectory synchronization processes can modify the information on this replica.
RT_SUBREF	3	Identifies the replica as a subordinate reference. eDirectory automatically adds these replicas to a server when the server does not contain replicas of all child partitions. Only eDirectory can modify information on this replica.

5.24 Replica States

The replica states indicate the current state of the replica. NetWare 4.x and NetWare 5.x, at times, use different states. For more information, see "Replica Transition States" (*NDK: Novell eDirectory Technical Overview*).

Flag Name	C Value	Meaning
RS_ON	0	Indicates that the replica is fully functioning and capable of responding to eDirectory requests.
RS_NEW_REPLICA	1	Indicates that a new replica has been added but has not received a full download of information from the master replica if NetWare 4.x or another replica if NetWare 5.x.
RS_DYING_REPLICA	2	Indicates that a replica of the partition is being deleted. In NetWare 4.x, the replica stays in this state until it synchronizes with another replica. In NetWare 5.x, indicates that the request has been received.
RS_LOCKED	3	Indicates that the replica is locked. The move partition operation uses this state to lock the parent partition of the child partition that is moving.
RS_CRT_0	4	Indicates that a partition is receiving a new master replica. This first state indicates that the old master replica has accepted the assignment to be a Read/Write replica. Used only by NetWare 4.x.
RS_CRT_1	5	Indicates that a Read/Write replica has been accepted as the new master replica and is now in the process of informing the other replicas. Once all replicas have been informed, the replicas will be set to RS_ON. Used only by NetWare 4.x.
RS_TRANSITION_ON	6	Indicates that a new replica has finished receiving its download from the master replica and is now receiving synchronization updates from the other replicas. Used only in NetWare 4.x.

Flag Name	C Value	Meaning
RS_DEAD_REPLICA	7	Indicates that the dying replica needs to synchronize with another replica before being converted to an external reference, if a root replica, or to a subordinate reference, if a nonroot replica. Used only in NetWare 5.x.
RS_BEGIN_ADD	8	Indicates that subordinate references of the new replica are being added. Used only in NetWare 5.x.
RS_MASTER_START	11	Indicates that a partition is receiving a new master replica. The replica that will be the new master replica is set to this state.
RS_MASTER_DONE	12	Indicates that a partition has a new master replica. When the new master is set to this state, it knows it is now the master and changes its replica type to master and the old master to Read/Write.
RS_FEDERATED	13	Reserved.
RS_SS_0	48	Indicates that a partition is going to split into two partitions. In this state, other replicas of the partition are informed of the pending split operation.
RS_SS_1	49	Indicates that the split partition operation has started. When the split is finished, the state will change to RS_ON.
RS_JS_0	64	Indicates that two partitions are in the process of joining into one partition. In this state, the replicas that are affected are informed of the join operation. The master replica of the parent and child partitions are first set to this state and then all the replicas of the parent and child. New replicas are added where needed.
RS_JS_1	65	Indicates that two partitions are in the process of joining into one partition. This state indicates that the join operation is waiting for the new replicas to synchronize and move to the RS_ON state.
RS_JS_2	66	Indicates that two partitions are in the process of joining into one partition. This state indicates that all the new replicas are in the RS_ON state and that the rest of the work can be completed.
RS_MS_0	80	Indicates that a subtree is being moved to another location in the eDirectory tree. In this state, the replicas that are affected are informed of the move request. The replicas of the new parent partition are first set to this state, and then the replicas of the subtree that is moving. In NetWare 4, all move operations are completed in this state. In NetWare 5, once all replicas have been informed of the move request, the replicas move to the next state.

Flag Name	C Value	Meaning
RS_MS_1	81	Indicates that a subtree is being moved to another location in the NDS tree and that all affected replicas have been notified. The replicas remain in this state until the move operations are completed. Used only in NetWare 5.x.

5.25 Syntax Matching Flags

The NWDSReadSyntaxDef (page 346) and NWDSGetSyntaxDef (page 238) functions use a Syntax_Info_T (page 457) structure to return information about syntax definitions. The structure uses the following flags to return information about the syntax's matching rules. These flags are ORed together when multiple flags apply to a syntax.

Flag	C Value	Description
DS_STRING	0x0001	Indicates that the syntax can contain string values and therefore attributes with this syntax can be used as naming attributes.
DS_SINGLE_VALUED	0x0002	Indicates that attributes using this syntax can have only one value.
DS_SUPPORTS_ORDER	0x0004	Indicates that attributes using this syntax must be open to comparisons of less than, equal to, and greater than.
DS_SUPPORTS_EQUAL	0x0008	Indicates that attributes using this syntax match for equality when all of the following conditions are met. The attributes' values are identical; the attributes use the same syntax, and the attributes' data type conforms to the syntax.
DS_IGNORE_CASE	0x0010	Indicates that attributes using this syntax ignore case during comparisons.
DS_IGNORE_SPACE	0x0020	Indicates that attributes using this syntax ignore extra space during comparisons.
DS_IGNORE_DASH	0x0040	Indicates that attributes using this syntax ignore dashes during comparisons.
DS_ONLY_DIGITS	0x0080	Indicates that attributes using this syntax must support only digits in their values.
DS_ONLY_PRINTABLE	0x0100	Indicates that attributes using this syntax must support only printable characters in their values. For a list of these characters, see the “Printable String” syntax.
DS_SIZEABLE	0x0200	Indicates that attributes using this syntax must set upper and lower limits to their values.
DS_BITWISE_EQUAL	0x0400	Indicates that attributes using this syntax support substring (wildcard) and approximate matching.

5.26 Syntax IDs

The Syntax IDs identify an attribute syntax definition (for more information, see "Attribute Syntax Definitions" (*NDK: Novell eDirectory Schema Reference*)). The IDs are defined in the nwdsdefs.h and nwdsdefs.inc files.

Name	C Value	Description
SYN_UNKNOWN	0	Stores values which are binary strings.
SYN_DIST_NAME	1	Stores values which are the names of objects in the eDirectory tree.
SYN_CE_STRING	2	Stores values which are Unicode strings, and these strings are case sensitive in comparison operations.
SYN_CI_STRING	3	Stores values which are Unicode strings, and these strings are case insensitive in comparison operations.
SYN_PR_STRING	4	Stores values which are printable strings as defined in CCITT X.208.
SYN_NU_STRING	5	Stores values which are numeric strings as defined in CCITT X.208.
SYN_CI_LIST	6	Stores values which are ordered sequences of Unicode strings, and these strings are case insensitive in comparison operations.
SYN_BOOLEAN	7	Stores values which are either TRUE or FALSE.
SYN_INTEGER	8	Stores values which are signed numeric integers.
SYN_OCTET_STRING	9	Stores values which are binary strings.
SYN_TEL_NUMBER	10	Stores values which are telephone numbers.
SYN_FAX_NUMBER	11	Stores values which are strings that comply with the format agreed upon for international telephone numbers (E.123) and optional bit strings (Recommendation T.30).
SYN_NET_ADDRESS	12	Stores values which represent network-layer addresses in binary format.
SYN_OCTET_LIST	13	Stores values which are ordered sequences of binary strings.
SYN_EMAIL_ADDRESS	14	Stores values which are strings of binary information.
SYN_PATH	15	Stores values which contain the file system path information for locating a file on a NetWare volume.
SYN_REPLICA_POINTER	16	Stores values which are used by Replica attributes.
SYN_OBJECT_ACL	17	Stores values which are ACL attributes.

Name	C Value	Description
SYN_PO_ADDRESS	18	Stores values which are Unicode strings of postal addresses.
SYN_TIMESTAMP	19	Stores values which mark the time when a particular event occurred.
SYN_CLASS_NAME	20	Stores values which are eDirectory object class names. These names are case sensitive in comparison operations.
SYN_STREAM	21	Stores values which are arbitrary binary information.
SYN_COUNTER	22	Stores values which are incrementally modified, signed integers. Modifications to the value are arithmetically added to, or subtracted from, the total.
SYN_BACK_LINK	23	Stores the identity of the object that requires an external reference and the server that stores the reference.
SYN_TIME	24	Stores values which are unsigned integers and which represent time in seconds.
SYN_TYPED_NAME	25	Stores values which have a level and an interval associated with an object name.
SYN_HOLD	26	Stores values which are signed integers, have a server associated with them, and are modified arithmetically with addition and subtraction.
SYN_INTERVAL	27	Stores values which are signed numeric integers and which also represent intervals of time.
SYNTAX_COUNT	28	Returns the number of syntax definitions

eDirectory Example Code

6

This chapter provides links to example code for the common tasks that most Novell® eDirectory™ applications require.

6.1 Context Handle

Every function has a parameter for a context handle. The NDSCONTX file shows how to create a context handle and displays some of its settings. For source code, see ndscontx.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/ndscontx/NDSCONTX.C.html).

6.2 Object and Attribute

The following examples show how to add, delete, and modify a single object and its attributes:

- NDSADD1—adds a User object. For source code, see ndsadd1.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/ndsadd1/NDSADD1.C.html).
- DELUSER—deletes a User object. For source code, see deluser.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/deluser/DELUSER.C.html).
- NDSMODOB—modifies an attribute of a User object. For source code, see ndsmodob.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/ndsmodob/NDSMODOB.C.html).

6.3 Browsing and Searching

The following examples show how to browse and search the eDirectory tree for various types of information:

- NDSBROWS—lists the objects in an eDirectory container. For source code, see ndsbrows.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/ndsbrows/NDSBROWS.C.html).
- LISTPART—lists the partitions on a specified server. For source code, see listpart.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/listpart/LISTPART.C.html).
- NDSREADA—reads attributes of a specified User object. For source code, see ndsreada.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/ndsreada/NDSREADA.C.html).
- READACL—reads the ACL attribute of an object. For source code, see readacl.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/readacl/readacl.c.html).

- READEFF—reads the rights one eDirectory object has to another eDirectory object. For source code, see readeff.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/readeff/readeff.c.html).
- READINFO—reads non-attribute information that eDirectory maintains about an object such as the object's base class, modification time, and entry flags that indicate such items as whether the object is a container, a partition, or an alias. For source code, see readinfo.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/readinfo/readinfo.c.html).
- NDSSEARCH—searches for objects of a specified class in a specified container and its subordinates. For source code, see ndssearc.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/ndssearc/NDSSEARC.C.html).

6.4 Batch Modification of Objects and Attributes

The following files work together to show how to read information from the eDirectory database and store it in a file, modify attribute information in the file, and then apply to the changes to the eDirectory database.

- DSEXTSNC—searches the database, returns the object and attributes that match the search filter, and writes the information to a file. For source code, see dsextsnc (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/syncds/dsextsnc.c.html).
- READDB—reads the information from the file, displays it so the user can change it, and saves the changes to the file. For source code, see readdb.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/syncds/readdb.c.html).
- DSINSYNC—reads the information in the file and modifies the eDirectory database to match the modifications in the file. For source code, see dsinsync.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/syncds/dsinsync.c.html).

6.5 Schema

The following examples show how to read, modify, and create schema definitions:

- CRCLSDEF—creates a new object class definition. For source code, see crclsdef (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/crclsdef/CRCLSDEF.C.html).
- RDATTDEF—reads the attribute definition of a specified attribute. For source code, see rdattdef (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/rdattdef/RDATTDEF.C.html).
- RDCLSDEF—lists all the object class definitions that exist in the schema. For source code, see rdclsdef.c (http://developer.novell.com/ndk/doc/samplecode/ndslib_sample/rdclsdef/RDCLSDEF.C.html).

Revision History

A

The following table lists all changes made to the eDirectory™ Core Services documentation:

June 2008	Fixed a typo in the Pascal function name of NWDSLoginEx (page 272).
February 2008	To accommodate 64-bit platforms, changed the data type of the iteration handle from pnint32 to pnint_ptr. The typedefs from ntypes.h handles this change for backwards compatibility with older 32-bit applications.
October 11, 2006	Added an explanation to NWIsDSAuthenticated (page 419) about it being obsolete and the functions to call instead. Also added the Remarks section that explains how the function works on NetWare® and Windows. Added note about 0 results being returned if none of the alternate replicas can be contacted to the Remarks section of NWDSSearch (page 383). Added that DS_SEARCH_PARTITION works with NDS 8 and later versions in Section 5.22, “Scope Flags,” on page 483. Updated the description of DSI_ENTRY_DN in Section 5.11, “DCK_DSI_FLAGS Values,” on page 472.
March 1, 2006	Added some clarification to the DCK_TREE_NAME description in Section 5.6, “Context Keys and Flags,” on page 467. Added navigational links.
October 5, 2005	Transitioned to revised Novell documentation standards.
October 2004	Added more information on NWDSVerifyPwdEx (page 406).
February 2004	Changed NDS to eDirectory.
March 2003	◆ Updated optionsFlag parameter of NWDSGenerateKeyPairEx (page 164) and NWDSChangeObjectPassword (page 116) with correct options. ◆ Clarified NWDSLogout (page 275) to state that the rigts associated with server attachments and session connections are lost while the connection remains intact.
May 2002	Updated the Pascal syntax of Syntax_Info_T (page 457).
September 2001	Updated the pascal syntax for the following functions: ◆ NWDSPutClassName (page 321) ◆ NWDSPutSyntaxName (page 325) ◆ NWDSScanConnsForTrees (page 379)
July 2000	Added information to the NWDSReadNDSInfo and NWDSGetDSVerInfo functions and clarified the use of the changeType flags for modifying objects.