Novell Developer Kit

www.novell.com

NOVELL® EDIRECTORY™ ITERATOR SERVICES

February 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.

Copyright © 1999-2008 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

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

AppNotes is a registered trademark of Novell, Inc.
AppTester is a registered trademark of Novell, Inc., in the United States.
ASM is a trademark of Novell, Inc.
BorderManager is a registered trademark of Novell, Inc.
BrainShare is a registered service mark of Novell, Inc., in the United States and other countries.
C3PO is a trademark of Novell, Inc.
Certified Novell Engineer is a service mark of Novell, Inc.
Client32 is a trademark of Novell, Inc.
CNE is a registered service mark of Novell, Inc.
ConsoleOne is a registered trademark of Novell, Inc.
Controlled Access Printer is a trademark of Novell, Inc.
Custom 3rd-Party Object is a trademark of Novell, Inc.
DeveloperNet is a registered trademark of Novell, Inc., in the United States and other countries.
DirXML is a registered trademark of Novell, Inc.
eDirectory is a trademark of Novell, Inc.
Excelerator is a trademark of Novell, Inc.
exteNd is a trademark of Novell, Inc.
exteNd Director is a trademark of Novell, Inc.
exteNd Workbench is a trademark of Novell, Inc.
FAN-OUT FAILOVER is a trademark of Novell, Inc.
GroupWise is a registered trademark of Novell, Inc., in the United States and other countries.
Hardware Specific Module is a trademark of Novell, Inc.
Hot Fix is a trademark of Novell, Inc.
iChain is a registered trademark of Novell, Inc.
Internetwork Packet Exchange is a trademark of Novell, Inc.
IPX is a trademark of Novell, Inc.
IPX/SPX is a trademark of Novell, Inc.
jBroker is a trademark of Novell, Inc.
Link Support Layer is a trademark of Novell, Inc.
LSL is a trademark of Novell, Inc.
ManageWise is a registered trademark of Novell, Inc., in the United States and other countries.
Mirrored Server Link is a trademark of Novell, Inc.
Mono is a registered trademark of Novell, Inc.
MSL is a trademark of Novell, Inc.
My World is a registered trademark of Novell, Inc. in the United States.
NCP is a trademark of Novell, Inc.
NDPS is a registered trademark of Novell, Inc.
NDS is a registered trademark of Novell, Inc., in the United States and other countries.
NDS Manager is a trademark of Novell, Inc.
NE2000 is a trademark of Novell, Inc.
NetMail is a registered trademark of Novell, Inc.
NetWare is a registered trademark of Novell, Inc., in the United States and other countries.
NetWare/IP is a trademark of Novell, Inc.
NetWare Core Protocol is a trademark of Novell, Inc.
NetWare Loadable Module is a trademark of Novell, Inc.

NetWare Management Portal is a trademark of Novell, Inc.
NetWare Name Service is a trademark of Novell, Inc.
NetWare Peripheral Architecture is a trademark of Novell, Inc.
NetWare Requester is a trademark of Novell, Inc.
NetWare SFT and NetWare SFT III are trademarks of Novell, Inc.
NetWare SQL is a trademark of Novell, Inc.
NetWire is a registered service mark of Novell, Inc., in the United States and other countries.
NLM is a trademark of Novell, Inc.
NMAS is a trademark of Novell, Inc.
NMS is a trademark of Novell, Inc.
Novell is a registered trademark of Novell, Inc., in the United States and other countries.
Novell Application Launcher is a trademark of Novell, Inc.
Novell Authorized Service Center is a service mark of Novell, Inc.
Novell Certificate Server is a trademark of Novell, Inc.
Novell Client is a trademark of Novell, Inc.
Novell Cluster Services is a trademark of Novell, Inc.
Novell Directory Services is a registered trademark of Novell, Inc.
Novell Distributed Print Services is a trademark of Novell, Inc.
Novell iFolder is a registered trademark of Novell, Inc.
Novell Labs is a trademark of Novell, Inc.
Novell SecretStore is a registered trademark of Novell, Inc.
Novell Security Attributes is a trademark of Novell, Inc.
Novell Storage Services is a trademark of Novell, Inc.
Novell, Yes, Tested & Approved logo is a trademark of Novell, Inc.
Nsure is a registered trademark of Novell, Inc.
Nterprise is a trademark of Novell, Inc.
Nterprise Branch Office is a trademark of Novell, Inc.
ODI is a trademark of Novell, Inc.
Open Data-Link Interface is a trademark of Novell, Inc.
Packet Burst is a trademark of Novell, Inc.
PartnerNet is a registered service mark of Novell, Inc., in the United States and other countries.
Printer Agent is a trademark of Novell, Inc.
QuickFinder is a trademark of Novell, Inc.
Red Box is a trademark of Novell, Inc.
Red Carpet is a registered trademark of Novell, Inc., in the United States and other countries.
Sequenced Packet Exchange is a trademark of Novell, Inc.
SFT and SFT III are trademarks of Novell, Inc.
SPX is a trademark of Novell, Inc.
Storage Management Services is a trademark of Novell, Inc.
SUSE is a registered trademark of SUSE AG, a Novell business.
System V is a trademark of Novell, Inc.
Topology Specific Module is a trademark of Novell, Inc.
Transaction Tracking System is a trademark of Novell, Inc.
TSM is a trademark of Novell, Inc.
TTS is a trademark of Novell, Inc.
Universal Component System is a registered trademark of Novell, Inc.

Virtual Loadable Module is a trademark of Novell, Inc.

VLM is a trademark of Novell, Inc.

Yes Certified is a trademark of Novell, Inc.

ZENworks is a registered trademark of Novell, Inc., in the United States and other countries.

Third-Party Materials

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

Java is a trademark or registered trademark of Sun Microsystems, Inc., in the United States and other countries.

Contents

About This Guide

Novell® eDirectory™ Iterator Services provide methods for searching and retrieving object information from eDirectory containers that have thousands or hundreds of thousands of objects. The functions work most efficiently on eDirectory servers who have upgraded to NDS 8. However, these functions do work on all NDS/eDirectory servers. Information about the iterator services has been 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 Backup Services (http://developer.novell.com/ndk/doc/ndslib/dsbk_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 Core Services (http://developer.novell.com/ndk/doc/ndslib/nds__enu/data/h2y7hdit.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/edirectory.html).

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.

Concepts

1

NDS® 8 is designed to handle hundreds of thousands of objects in a single container. To handle such large numbers of objects, the NDS/Novell® eDirectory™ database has per-server indexes for the objects which sort the objects by the object's RDN, given name, and surname, to name a few. The eDirectory iterator functions access these indexes and allow developers to position an iterator object on entries in the index and to retrieve information about groups of eDirectory objects. The iterator functions can also be used on eDirectory servers running earlier versions of NDS.

The following sections introduce you to the iterator objects and the virtual lists they reference, explain how the functions position the iterator in the list, and describe how to extract data from the list.

1.1 Iterator Objects

An iterator object is the object created to access a virtual list of eDirectory objects. You use this iterator object to move around in the list and to retrieve data from the list. It provides list and search functionality similar to the NWDSList and NWDSSearch functions, but can deal with very large result sets.

For example, an application can display a scrollable listbox of 20 names from a large alphabetized virtual list of 100,000 eDirectory users. Because the server keeps the list in sorted order, the application needs to read only a small group of names for display at a time.

An application may also want to search a large eDirectory database for all users with the given name "Mary." NDS 8 allows this search to be done very efficiently, without having to examine every object in the database.

1.2 Creation of an Iterator Object

An iterator is created with one of the following functions:

- The NWDSItrCreateList function specifies many of the same parameters used in the NWDSList function. It is simple to use, but has limited functionality. It works best when enumerating objects in a container.
- The NWDSItrCreateSearch function specifies many of the same parameters used in the NWDSSearch function. It is more complex to use than the NWDSItrCreateList function, but has much greater functionality. It has a search filter to indicate which objects to return. Other parameters specify how much information to return on the objects which pass the search filter

For information on search filter syntax, see "Search Filter Components" (*NDK: Novell eDirectory Core Services*). For step-by step instructions on creating an iterator, see Section 2.1, "Creating a Search Iterator Object," on page 21.

An iterator can be cloned to produce a copy of itself, which may be positioned independently of the original. All iterators created with the NWDSItrCreateSearch, NWDSItrCreateList, or NWDSItrClone functions must be destroyed with the NWDSItrDestroy function. This function frees the resources associated with the iterator on both clients and servers.

Conceptually, an iterator creates a virtual list view (VLV) on the server containing all the entries that pass the search filter or list parameters. This list contains only objects on the particular server in question. An iterator can do a subtree search of objects in the replicas on the server, but the subtree search does not span servers.

1.3 Iterator Indexes

NDS 8 creates multiple indexes for the objects that are found in the replicas stored on the server. These indexes sort objects based on their attributes. For example, one index sorts the objects by base class (object class used to create the object) and then alphabetically by the object's RDN.

The eDirectory libraries emulate the index functionality on eDirectory servers not running NDS 8. Hence, a number of differences exist between how true indexes and emulated indexes work.

The information about index features has been divided into the following topics:

1.3.1 Types

The eDirectory iterator object can be configured to use one of the following indexes.

"_BaseClass, _RDN"	Sorts objects by their base class, and then by their naming attribute.
"Surname"	Sorts User objects by their Surname attribute.
"Given Name"	Sorts User objects by their Given Name attribute.
"CN"	Sorts objects by their CN attribute. Most leaf objects use the CN attribute for their naming attributes, whereas most container objects use other attributes for naming.
"UniqueID"	Sorts objects by their uniqueID attribute (LDAP's uid attribute). This is not a required eDirectory attribute. Unless a utility has been run that creates these for the objects in the eDirectory tree, this index may contain no entries.

Two of the attributes, "CN" and "UniqueID," contain substring indexes on their values. Substring indexes contain multiple keys for each value so that searching for a specific substring of a value is optimized. For example, a search string of

```
*ary
```

would quickly find the values

```
Hillary
Mary
Scary
```

The index can find these values without comparing the substring value of "ary" with every string in the index.

1.3.2 Positionable and Nonpositionable Iterators

A positionable iterator is an iterator object that has information about where an entry is positioned in the list. A positionable iterator can be positioned with the NWDSItrSetPosition function and can return its position with the NWDSItrGetPosition function. Iterators that are nonpositionable can return information about entries but can't return information about an entry's location in the list.

Whether an iterator is positionable depends upon the interaction between the search filter and the index (see “Search Filters” on page 13). The NWDSItrGetInfo function returns whether an existing iterator is positionable.

1.3.3 Search Filters

The objects returned by the iterators can be restricted by applying a search filter. The NWDSItrCreateList function uses two parameters for its search filter:

- className can be used to restrict the list to objects of a specified base class. Wildcards cannot be used to specify the base class.
- subordinateName can be used to restrict the list to objects that match a specified RDN. This name can include wildcards.

The NWDSItrCreateSearch function uses the same type of search filter as the NWDSSearch function, with a few restrictions on attributes and wildcards. For complete information on creating a search filter, see “Search Requests” (*NDK: Novell eDirectory Core Services*).

Object Class Attribute. This attribute should not be used in the search filter. This attribute is a multivalued attribute that contains the object's base class and all its super classes. For example, the Object Class attribute for User contains the following objects: User, Organizational Person, Person, and Top. A search filter of "Object Class=User" is not efficient since eDirectory must search through multiple values in the attribute for every object in the container. The "BaseClass" string allows for an efficient search of an object's base class and should be used to restrict a search to a specific base class.

Other eDirectory Attributes. For positionable iterators, an attribute should not be included in the search filter unless it is an attribute of the index. The following table lists the indexes and the attributes that can be used to produce a positionable iterator.

Index	Attribute	Search Filter Expression
"_BaseClass, _RDN"	BaseClassRDN	"BaseClass=User" or"RDN=S*" or"BaseClass=User AND RDN=S*"
"CN"	CN	"CN=D*"
"Given Name"	Given Name	"Given Name=S*"
"Surname"	Surname	"Surname=B*"
"UniqueID"	uniqueID	"uniqueID=A*"

For nonpositionable iterators, any eDirectory attribute can be included in the filter. For example, the following filter produces a nonpositionable iterator:

```
"BaseClass=User AND Telephone Number=571*"
```

Also, ANDing the same attribute together with multiple values produces a nonpositionable iterator. For example, the following filter produces a nonpositionable iterator:

```
"CN=J* AND CN=S*"
```

eDirectory sometimes appends criteria to the search request. For example, the criteria might ensure that the selected entries are not in a deleted state or that the results are limited to a particular container in the eDirectory tree. These criteria can cause an iterator, which meets the other rules for positionable, to be nonpositionable.

If your application requires a positionable iterator, use the NWDSItrGetInfo function to verify that functionality.

Wildcards. Wildcards can be used in any text attribute except for BaseClass. Wildcards can be used in any combination. Search filters ending in a single wildcard are very efficient with any index. Leading wildcard searches, such as "*sen" depend on the index and the size of the container. On large containers with nonsubstring indexes ("_BaseClass, _RDN", "Given Name", and "Surname"), leading wildcard searches do not produce an efficient search query.

On substring indexes ("CN" and "UniqueID"), leading wildcard searches are efficient. However, the sort order is unpredictable. For example, a search filter of CN=*a* produces a random list of names that contain the letter, such as Jane, David, and Anna. Also, such an iterator object is not positionable.

RDN String. The schema uses multiple naming attributes (for example, CN, L, OU, or O) for the object classes. The RDN string specifies an object's name and uses the FTOK_RDN token in the search filter. The search expression "RDN=S*" uses the following node expressions in constructing a search filter:

```
FTOK_RDN, NULL, 0
FTOK_ANAME, "S*", SYN_DIST_NAME
```

BaseClass String. The BaseClass string uses the FTOK_BASECLS token in the search filter. The search expression "BaseClass=User" uses the following node expressions:

```
FTOK_BASECLS, NULL, 0
FTOK_ANAME "User", SYN_CLASS_NAME
```

1.3.4 Duplicate Entries

The eDirectory schema has many multivalued attributes. The only index using such an attribute is the "CN" index. The eDirectory search on an NDS 8 server returns duplicate entries when a multivalued attribute has more than one value. Since CN is such an attribute and most leaf objects in an eDirectory tree use CN as their naming attribute, most leaf objects can have more than one name. The indexes contain an entry for each name. For example, if a User object has Angela as its first CN value with additional values of Mary and Zella, the object appears three times in the list, once with the A's, once with the M's, and once with the Z's. In addition, since entry display is by first value, most applications would display Angela in all three locations.

1.3.5 Emulation Mode

True iterators require NDS 8 servers, which automatically create indexes for every container object in the replicas that they store. Thus, when a request comes in to create an iterator for a container, the server already has a sorted, indexed list to use for the iterator.

On eDirectory servers not running NDS 8, the libraries emulate this sorting and indexing functionality so that applications using these APIs work in both environments. If an iterator is created in the emulation mode, the workstation retrieves the entire result set, sorts it, and stores it in memory. The creation function thus takes longer and is not advised for very large lists.

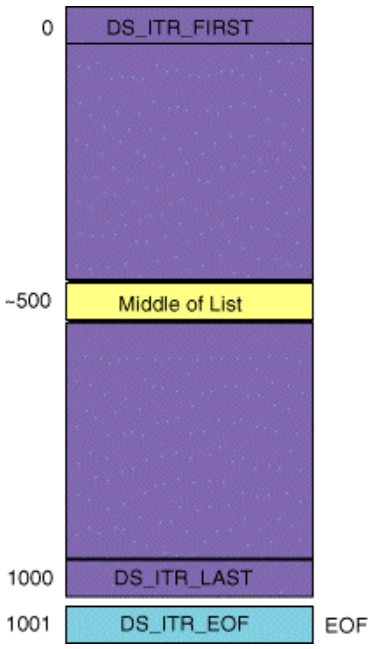
An iterator with an emulated index has the following characteristics:

- It is always positionable.
- The sortKey parameter can use any eDirectory attribute as well as the defined BaseClass and RDN strings.
- The search filter can use any eDirectory attribute.
- Duplicate entries are not returned. Only the first value of an attribute is added to the sorted list.

1.4 Positions of an Iterator Object

When the iterator object is created, its current position is set to the first object in the list that matches the search criteria. The figure below illustrates a virtual list and some of the possible positions for the iterator object.

Figure 1-1 *Virtual List with Iterator Object Possible Positions*



The numbers (0, ~500, 1000, and 1001) on the left side of the list represent logical positions in the list, not actual entry numbers. A logical position is a scaled number from 0 to 1000 and not an exact index number into the list. For example, 500 represents the approximate middle of the list, whether there are actually 50 or 5,000 entries in the list.

Three positions are exact:

- Logical position DS_ITR_FIRST (0) always corresponds to the first entry.
- DS_ITR_LAST (1000) always corresponds to the last entry in the list.
- DS_ITR_EOF (1001) indicates the end of list position, one beyond the last entry.

The position can be set to any entry in the list, or to EOF (end of file). EOF is an imaginary position after the end of the last entry. The following functions set the position or return information about the current position.

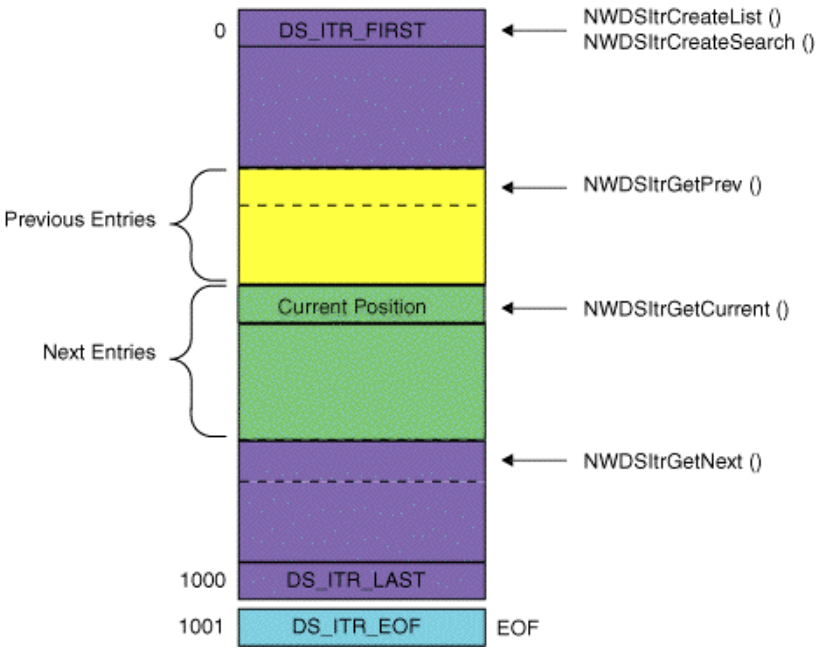
Function	Purpose
NWDSItrAtEOF	Indicates whether the iterator object is positioned at the end of the list. Works on all iterators.
NWDSItrAtFirst	Indicates whether the iterator object is at the first entry of the list. Works on all iterators.
NWDSItrGetPosition	Returns the current position of the iterator object. Works on positionable iterators.
NWDSItrSetPosition	Moves the current position to the specified position. Logical values specify the location: 0, the top of the list; 1000, the end of the list; and 1001, the EOF position. Other values move the current position to an approximate location. Works on positionable iterators.

Function	Purpose
NWDSItrSkip	Allows the current position to be moved forward or backward “n” entries. Skipping beyond the top of the list leaves the current position at the first entry in the list (DS_ITR_FIRST). Skipping beyond the end of the list leaves the position at the end of the list (DS_ITR_EOF). Works on all iterators.
NWDSItrSetPositionFromIterator	Moves the current position of the iterator to the object that comes the closest to matching the current object of the source iterator object. For NDS 8 iterators, the iterators do not need to be identical. For emulation mode, the iterators need to be clones.
NWDSItrTypeDown	Moves the current position to the object with the attribute that matches the input string. For example, a typedown using attribute “Surname” and string “St” positions the iterator to the first entry with surname “St”. If no entries starting with “St” are present, the iterator is positioned on the first entry after “St” in sorted order. Works on all iterators.

1.5 Current Position Movement with Retrieval Functions

Upon creation, the iterator's current position is set to the first entry in the list. Thereafter, the current position changes as object data is retrieved. The figure below shows a virtual list and the various positions that the retrieval functions move the iterator's current position.

Figure 1-2 *Virtual List*



The NWDSItrCreateList and NWDSItrCreateSearch functions position the iterator on the first object in the list. The NWDSItrGetCurrent function retrieves the data of the object in the current position; it does not move the iterator's position.

The current position moves with the other data retrieval functions. The NWDSItrGetNext function retrieves the information about the n number of entries specified, including the current entry. It then positions the iterator on the next available entry. Notice that the arrow in the figure points to an entry beyond the “Next Entries” box.

The NWDSItrGetPrev function retrieves the information about n number of entries specified, in reverse sort order (4, 3, 2, 1) from the current position. It does not retrieve information about the object in the current position when the function is called, and it positions the iterator on the entry last read. Notice that the arrow in the figure points to the last entry in the “Previous Entries” box.

1.6 Retrieval of Data

The data retrieval functions return entries to the user relative to the current position.

Function	Description
NWDSItrGetCurrent	Returns the entry at the current position, leaving the position unchanged.
NWDSItrGetNext	Returns “n” entries starting from the current position, including the current entry. The position is moved to the next unread entry.
NWDSItrGetPrev	Returns the previous “n” entries in reverse sort order, not including the current entry. The position is moved to the last entry read, which is closest to the top of the list.

The following sections describe how these functions work and their common characteristics. These include

- Section 1.6.1, “Iteration Handles and Result Buffers,” on page 18
- Section 1.6.2, “Data Unpacking Functions,” on page 19
- Section 1.6.3, “Count of Objects in the List,” on page 19

1.6.1 Iteration Handles and Result Buffers

All data retrieval functions have an iteration handle parameter and a result buffer parameter. The iteration handle is used to retrieve additional information when the requested information does not fit in the result buffer. The default buffer size is 4 KB (DEFAULT_MESSAGE_LEN). However, you can set it to a smaller size or to a larger size, depending the types of data your application uses. The maximum size is 63 KB (MAX_MESSAGE_LEN).

You must set the iteration handle to -1 the first time you call a “Get” function. If the iteration handle is still set to -1 after the call, all the data has been retrieved. If it has any other value, all of the data could not fit in the result buffer. You should unpack the data from the buffer and then repeatedly call the function to retrieve additional data until the iteration handle is set to -1.

1.6.2 Data Unpacking Functions

Information is retrieved from the result buffers with the same specialized functions that are used to retrieve information from the result buffers of the NWDSRead and NWDSSearch functions. Since the NWDSItrCreateSearch function supports only two information types (attribute names and attribute names with values), the iterator functions do not need as many specialized functions as NWDSRead to retrieve data. The data retrieval functions (NWDSItrGetCurrent, NWDSItrGetNext, and NWDSItrGetPrev) require the following functions to unpack the data from the result buffer.

Function	Description
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 the attribute's name and the number of values associated with the attribute. If values are not requested in the information type, the number of values is always zero.
NWDSGetAttrVal	Returns the attribute's value. This function must be called for each value associated with the attribute.
NWDSComputeAttrValSize	Returns the size of the attribute's value. This function is required only if you don't know the size of the attribute.

For step-by-step instructions for retrieving data, see Section 2.2, “Retrieving and Unpacking Object and Attribute Name Data,” on page 21 and Section 2.3, “Retrieving and Unpacking Object, Attribute, and Value Data,” on page 22.

1.6.3 Count of Objects in the List

The NWDSItrCount function returns the number of entries left in the list, starting from the current position. In very large lists, this might take a long time. A timeout value in seconds and a maxCount parameter might be supplied to limit the time NWDSItrCount can take, which allows for estimation of the total count. For example, starting from the top of the list, a 2-second count might return a timeout status and a count of 10,000 entries. Calling NWDSGetPosition might indicate the position is now at the 20% mark. Therefore, a reasonable estimate for the total count is 50,000. Specifying zero for both timeout and maxCount will return an exact count no matter how long it takes.

Tasks

2

This chapter provides step-by-step instructions for creating an iterator object, retrieving data, and unpacking the information.

2.1 Creating a Search Iterator Object

The following procedure describes how to create an iterator object that has a search filter and that specifies selected attributes.

1. Call NWDSAllocFilter to allocate a filter expression tree.
2. Call NWDSAddFilterToken once for each search token to place the search-filter conditions in the expression tree.

   Objects that do not meet the conditions of the search filter will not appear in the virtual list. For more information on search filters and search tokens, see “Search Filter Components” (*“NDK: Novell eDirectory Core Services”*).
3. Call NWDSAllocBuf to allocate a filter buffer.
4. Call NWDSPutFilter to store the search-filter expression tree in the filter buffer.
5. To create a list with selected attributes, call NWDSAllocBuf to allocate the request buffer.
   - To read all attributes, set the allAttrs parameter to TRUE and skip to Step 9.
   - To read no attribute information, set the allAttrs parameter to NULL and skip to Step 8.

   .
6. Call NWDSInitBuf to initialize the request buffer for a DSV_SEARCH operation.
7. Call NWDSPutAttrName once for each attribute name to place the names of the desired attributes into the request buffer.

   This attribute filter restricts the virtual list to those objects that have the attributes contained in this list of attributes. Objects that do not have these attributes will not appear in the virtual list.
8. Set the allAttrs parameter of the NWDSItrCreateSearch (page 38) function to FALSE.
9. Call NWDSItrCreateSearch to create the iterator object.

For sample code, see Section 4.4, “Creating a Search Iterator and Displaying the Results: Example,” on page 69.

2.2 Retrieving and Unpacking Object and Attribute Name Data

The following procedure explains how to retrieve data from a call to NWDSItrGetNext or NWDSItrGetPrev function when the information type of the iterator object consists of attribute

names without values. If your result buffer is not big enough to hold all the requested data, you must retrieve the data and then repeat the call to retrieve the next set of data.

1. Call NWDSAllocBuf to allocate the result buffer, which does not need to be initialized.
2. Call either NWDSItrGetNext (page 48) or NWDSItrGetPrev (page 52).
3. Call NWDSGetObjectCount to determine the number of objects whose information is stored in the buffer.
4. Call NWDSGetObjectName to get the name of the current object in the buffer and the count of attributes associated with the object.
5. Call NWDSGetAttrName to retrieve the name of the attribute and the count of values associated with the attribute. For this information type, the number of values will always be zero.
6. Repeat Step 5 until all attribute information for the object has been read.
7. Loop to Step 4 until the information for all objects in the buffer has been retrieved.
8. If the result buffer does not contain all the requested data, loop to Step 2 and call the request function again (either NWDSItrGetNext or NWDSItrGetPrev).

   If the result buffer contains all the requested data, continue with Step 9.
9. Call NWDSFreeBuf to free the result buffer.
10. Call NWDSItrDestroy (page 43) to destroy the iterator object.

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

See Also

- Section 2.3, “Retrieving and Unpacking Object, Attribute, and Value Data,” on page 22
- Section 4.4, “Creating a Search Iterator and Displaying the Results: Example,” on page 69

2.3 Retrieving and Unpacking Object, Attribute, and Value Data

The following procedure explains how to retrieve data from a call to NWDSItrGetNext or NWDSItrGetPrev function when the information type of the iterator object consists of attribute names and values. If your result buffer is not big enough to hold all the requested data, you must retrieve the data and then repeat the call to retrieve the next set of data.

1. Call NWDSAllocBuf to allocate the result buffer, which does not need to be initialized.
2. Call NWDSItrGetNext (page 48) or NWDSItrGetPrev (page 52).
3. Call NWDSGetObjectCount to determine the number of objects whose information is stored in the buffer.
4. Call NWDSGetObjectName to get the name of the current object in the buffer and the count of attributes associated with the object.
5. Call NWDSGetAttrName to retrieve the name of the attribute and the count of values associated with the attribute.
6. For each value associated with the attribute, call NWDSGetAttrVal to retrieve the value.
7. Loop through Steps 5 and 6 until all attribute information for the object has been read.

8 Loop to Step 4 until the information for all objects in the buffer has been retrieved.

9 If the result buffer does not contain all the requested data, loop to Step 2 and call the request function again (either NWDSItrGetNext or NWDSItrGetPrev).

   If the result buffer contains all the requested data, continue with Step 10.

10 Call NWDSFreeBuf to free the result buffer.

11 Call NWDSItrDestroy (page 43) to destroy the iterator object.

You must pull all information from the result buffer even if you do not plan to use it. There is no way to skip data.

See Also

- Section 2.2, “Retrieving and Unpacking Object and Attribute Name Data,” on page 21
- Section 4.4, “Creating a Search Iterator and Displaying the Results: Example,” on page 69

Functions

3

This chapter lists alphabetically the functions that can be used to create, destroy, and use iterators.

NWDSItrAtEOF

Determines whether the specified iterator object is positioned at the end of the list.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

nbool8  NWDSItrAtEOF (
   nuint_ptr    Iterator);
```

Parameters

Iterator

(IN) Specifies the iterator object to determine whether it is positioned at the end of the list.

Return Values

N_TRUE	Indicates that the iterator object is at the end of the list or that the list is empty.
N_FALSE	Indicates the iterator object is not at the end of the list.

Remarks

This function returns a boolean value.

This function works on all iterator objects, including those that are not positionable.

For sample code, see Section 4.5, “Getting Iterator Information: Example,” on page 73.

NCP Calls

0x2222 104 02 Send Novell® eDirectory™ Fragmented Request/Reply

See Also

NWDSItrAtFirst (page 28), NWDSItrGetPosition (page 50)

NWDSItrAtFirst

Determines whether the specified iterator object is positioned at the beginning of the list.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

nbool8  NWDSItrAtFirst (
   nuint_ptr    Iterator);
```

Parameters

Iterator

(IN) Specifies the iterator object to determine whether it is positioned at the beginning of the list

Return Values

N_TRUE	Indicates that the iterator object is at the beginning of the list
N_FALSE	Indicates that the iterator object is not at the beginning of the list or that the list is empty.

Remarks

This function returns a boolean value.

This function works on all iterator objects, including those that are not positionable.

For sample code, see Section 4.5, “Getting Iterator Information: Example,” on page 73.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrAtEOF (page 26), NWDSItrGetPosition (page 50)

NWDSItrClone

Creates a copy of the iterator.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrClone (
   nuint_ptr    Iterator,
   pnuint_ptr    pNewIterator);
```

Parameters

Iterator

(IN) Specifies the iterator object to copy.

pNewIterator

(OUT) Points to the new iterator object.

Return Values

0x0000 0000	SUCCESSFUL
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (-001 to -799).

Remarks

This function creates a second iterator with all the characteristics of the original iterator. When cleaning up, you must destroy both the original and the new iterator.

For sample code, see Section 4.1, “Cloning an Iterator Object: Example,” on page 63.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrCreateSearch (page 38)

NWDSItrCount

Returns the number of entries left in this iterator.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrCount (
   nuint_ptr  Iterator,
   nuint32    timeout,
   nuint32    maxCount,
   nbool8     updatePosition,
   pnuint32   pCount);
```

Parameters

Iterator

(IN) Specifies the iterator object.

timeout

(IN) Specifies the time (in milliseconds) allowed before returning an error. If 0 is passed, there is no time limit.

maxCount

(IN) Specifies how many entries to scan before stopping. If 0 is passed, there is no maximum limit.

updatePosition

(IN) Specifies whether the current position will be left pointing to the last entry counted.

N_TRUE The current position equals the last entry counted or EOF.
N_FALSE The current position is unchanged.

pCount

(OUT) Points to the number of entries counted.

Return Values

0x0000 0000	Indicates the function successfully counted all remaining entries in the list and reached the end of the list. The pCount parameter contains the actual number of objects counted.
ERR_ITR_MAX_COUNT	Indicates the function reached the maximum count.
ERR_QUERY_TIMEOUT	Indicates the function reached its timeout before counting all the entries. The pCount parameter contains the actual number of objects counted.
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (-001 to -799).

Remarks

NWDSItrCount will scan the list until one of the following occurs:

- The end of the list is reached (returns 0)
- maxCount entries have been counted (returns ERR_ITR_MAX_COUNT)
- The time limit expires (returns ERR_QUERY_TIMEOUT)

Entries are scanned starting from the current position. Make sure the current position is at the top of the list if you want a total count.

If both the timeout and maxCount parameters are zero, the function will scan the entire list and return an exact count. On NDS 8 servers, this process may be very slow on large lists. On eDirectory servers with previous versions of NDS, the timeout parameter is ignored since the count can be determined immediately.

If scanning stops due to reaching the values specified by timeout or maxCount, call NWDSItrGetPosition to estimate the real count. If more accuracy is needed, call NWDSItrCount again with a larger timeout or maxCount value.

If, on an NDS 8 server, you need only an approximate count of the number of entries in the list, use the following procedure.

1. Position the iterator at the top of the list with NWDSItrSetPosition.
2. Call NWDSItrCount to count for a specified number of entries and have the updatePosition parameter move to the last entry counted.
3. Call the NWDSItrGetPosition function to discover the iterator's current position.

By comparing the logical position with the number of entries you requested to count, you can estimate the total length of the list. For example, if you requested to count 5,000 objects and the current logical position is 500 (or 50%), you can estimate that there are 10,000 objects in the list.

For sample code, see Section 4.2, "Counting with eDirectory Iterators: Example," on page 65.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrGetPosition (page 50)

NWDSItrCreateList

Creates a list iterator object and returns a pointer to it for subsequent function calls.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrCreateList (
    NWDSContextHandle   context,
    pnstr8              baseObjectName,
    pnstr8              className,
    pnstr8              subordinateName,
    nuint32             scalability,
    nuint32             timeout,
    pnuint_ptr          pIterator);
```

Parameters

context

(IN) Specifies the context to use for the request.

baseObjectName

(IN) Specifies the starting object to use for the search (usually a container object).

className

(IN) Specifies an optional filter to restrict the list to the specified object class. It may be NULL to include all classes in the list.

subordinateName

(IN) Specifies an optional filter to restrict the list to objects that match the specified RDN. The RDN may include wildcards. It may be NULL to include all names in the list.

scalability

(IN) Specifies whether the function can connect to servers running version of NDS other than NDS 8.

timeout

(IN) Specifies the time (in milliseconds) for completing the command.

pIterator

(OUT) Points to the iterator object returned to the caller.

Return Values

0x0000 0000	SUCCESSFUL
ERR_QUERY_TIMEOUT	Indicates the function reached its timeout limit before it could create the iterator object
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (-001 to -799).

Remarks

The iterator object returned in the pIterator parameter is used in all other iterator APIs. When the application is done with the iterator object, the iterator object must be destroyed by calling the NWDSItrDestroy function.

This function is similar to the NWDSListByClassAndName function. The big difference is that the list is sorted.

NOTE: The iterator creation process on an NDS 8 server is quick because the container indexes are already created. On servers with previous versions of NDS, the creation process can be slow. Since the index is not already created, the process includes building a list of all the objects in the container and then sorting them. If there are thousands of objects in the container, the task can take such a long time to complete that the application, making the call, appears to malfunction or hang.

The baseObjectName parameter identifies the object (or possibly the root) to which the list 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. The information returned is affected by the context handle flags. For more information, see "Context Keys and Flags" (*"NDK: Novell eDirectory Core Services"*).

The scalability parameter uses the following values:

DS_ITR_ANY_SERVER	Connects to the closest available server with a replica containing the specified base object.
DS_ITR_REQUIRE_SCALABLE	Returns an error if unable to connect to a server running NDS 8.
DS_ITR_FORCE_EMULATION	Forces the eDirectory libraries to use the emulation mode even if connected to a server running NDS 8.

In emulation mode, the timeout parameter is used to measure the time for reading the data and does not include the time for resolving the name.

NOTE: Don't confuse the iterator object with the iteration handle. For more information, see Section 1.1, "Iterator Objects," on page 11.

For sample code, see Section 4.3, "Creating and Using a List Iterator: Example," 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

NWDSItrClone (page 30), NWDSItrDestroy (page 43), NWDSItrCreateSearch (page 38), NWDSListByClassAndName

NWDSItrCreateSearch

Creates an iterator object and returns a pointer to it for subsequent function calls.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrCreateSearch (
    NWDSContextHandle   context,
    pnstr8              baseObjectName,
    nint                scope,
    nbool8              searchAliases,
    pBuf_T              filter,
    pTimeStamp_T        pTimeFilter,
    nuint32             infoType,
    nbool8              allAttrs,
    pBuf_T              attrNames,
    pnstr8              indexSelect,
    pnstr8              sortKey,
    nuint32             scalability,
    nuint32             timeout,
    pnuint_ptr          pIterator);
```

Parameters

context

(IN) Specifies the context to use for the request.

baseObjectName

(IN) Specifies the starting object to use for the search (usually a container object).

scope

(IN) Specifies a base object only, its immediate subordinates, or a subtree search (see "Scope Flags" (*"NDK: Novell eDirectory Core Services"*)).

searchAliases

(IN) Specifies whether subordinate alias objects are dereferenced:

TRUE Dereference subordinate aliases

FALSE Do not dereference subordinate aliases

filter

(IN) Specifies the search filter.

pTimeFilter

(IN) Points to a "TimeStamp_T" structure containing a modification time stamp to filter with (optional).

infoType

(IN) Specifies whether attribute names or attribute names and values should be returned (see "Information Types for Search and Read" (*NDK: Novell eDirectory Core Services*)).

allAttrs

(IN) Specifies if all attributes should be returned:

TRUE All attributes should be returned

FALSE Only the attributes specified by attrNames should be returned

attrNames

(IN) Specifies a list of attributes to return if allAttrs is FALSE. Passing NULL specifies no attributes will be returned.

indexSelect

(IN) Specifies an optional string that selects the index that the iterator object can use. Used only on servers running NDS 8.

sortKey

(IN) Specifies a comma-separated list of attributes that are used to sort the objects in the list. Used only on servers not running NDS 8.

scalability

(IN) Specifies whether the function can connect to servers running version of NDS other than NDS 8.

timeout

(IN) Specifies the time (in milliseconds) for completing the command.

pIterator

(OUT) Points to the iterator object returned to the caller.

Return Values

0x0000 0000	SUCCESSFUL
ERR_QUERY_TIMEOUT	Indicates the function reached its timeout limit before it could create the iterator object
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (-001 to -799).

Remarks

The iterator object returned in the pIterator parameter is used in all other iterator APIs. When the application is done with the iterator object, the iterator object must be destroyed by calling the NWDSItrDestroy function.

NOTE: The iterator creation process on an NDS 8 server is quick because the container indexes are already created. On servers with previous versions of NDS, the creation process can be slow. Since the index is not already created, the process includes building a list of all the objects in the container and then sorting them. If there are thousands of objects in the container, the task can take such a long time to complete that the application, making the call, appears to malfunction or hang.

The baseObjectName parameter identifies the object (or possibly the root) to which the list 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. The information returned is affected by the context handle flags. For more information, see "Context Keys and Flags" (*"NDK: Novell eDirectory Core Services"*).

Subtree searches are partially supported. If a subtree search is attempted by setting the scope parameter to DS_SEARCH_SUBTREE, the function searches the server's replicas. If the search requires replicas that do not exist on the server, the search returns with the information its has gathered from the server's replicas.

The filter parameter eliminates objects not of interest to the application. The search filter is created in the same manner as the search filter is created for the NWDSSearch function. It allows you to restrict the objects in the list to those objects that match the following types of conditions:

- Contain specified attributes
- Contain attributes with values equal to, greater than, or less than the specified value
- Belong to the same base class
- Have attributes with the same modification or creation timestamp

See "Search Requests" (*NDK: Novell eDirectory Schema Reference*) for more information on how to create a search filter.

If an iterator is created with a search filter that produces an empty list, the NWDSItrCreateSearch function succeeds. A list is produced with only the EOF position. Any read or move commands will return ERR_EOF_HIT. The iterator should be destroyed the same as any iterator.

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

If the allAttrs parameter is TRUE, information about all attributes associated with the object is requested and the attrNames parameter is ignored (in which case, the attrNames parameter can be NULL). If the allAttrs parameter is FALSE, only the attributes specified by the attrNames parameter are requested.

If the allAttrs parameter is FALSE and the attrNames parameter is NULL, no attribute information is returned and the infoType parameter is not meaningful.

If the indexSelect parameter is NULL or an empty string, the search filter is used to select the index on NDS 8 servers. If NULL is specified, eDirectory chooses the optimum index to use based on the search filter. To select the index, use one of the following strings:

"_BaseClass, _RDN"	Sorts objects by their base class, and then by their naming attribute.
"Surname"	Sorts User objects by their Surname attributes.
"Given Name"	Sorts User objects by their Given Name attributes.
"CN"	Sorts objects by their CN attribute. Most leaf objects use the CN attribute for their naming attributes, whereas most container objects use other attributes for naming.
"UniqueID"	Sorts objects by the LDAP uid attribute.

The sortKey parameter is ignored for NDS 8 servers. For eDirectory servers not running NDS 8, this parameter specifies an attribute string to use to sort the objects. The attribute must have a string syntax. If NULL is passed, "_BaseClass,_RDN" is used. This string is not an attribute but a special symbol that causes the iterator to be sorted first by object's base class, and within the base class, by the object's naming attribute. Up to three attributes can be specified, separated by a comma (for example, "Surname, Given Name, CN"). The first attribute in the list is considered the primary sort key.

The scalability parameter uses the following values:

DS_ITR_ANY_SERVER	Connects to the closest available server with a replica containing the specified base object.
DS_ITR_REQUIRE_SCALABLE	Returns an error if unable to connect to a server running NDS 8.
DS_ITR_FORCE_EMULATION	Forces the eDirectory libraries to use the emulation mode even if connected to a server running NDS 8. Useful for testing and performance comparisons.

In emulation mode, the timeout parameter is used to measure the time for reading the data and does not include the time for resolving the name.

NOTE: Don’t confuse the iterator object with the iteration handle. For more information, see Section 1.1, “Iterator Objects,” on page 11.

For sample code, see Section 4.4, “Creating a Search Iterator and Displaying the Results: Example,” 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

NWDSItrClone (page 30), NWDSItrDestroy (page 43), NWDSItrCreateList (page 35), NWDSSearch (*"NDK: Novell eDirectory Core Services"*)

NWDSItrDestroy

Destroys the iterator object and frees all associated memory.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrDestroy (
   nuint_ptr   Iterator);
```

Parameters

Iterator

(IN) Specifies the iterator object.

Return Values

0x0000 0000	SUCCESSFUL
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (-001 to -799).

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrCreateSearch (page 38), NWDSItrCreateList (page 35), NWDSItrClone (page 30)

NWDSItrGetCurrent

Returns the entry at the current position. The current position is left unchanged.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrGetCurrent (
   nuint_ptr    Iterator,
   pnint32      pIterationHandle,
   pBuf_T       pData);
```

Parameters

Iterator

(IN) Specifies the iterator object.

pIterationHandle

(IN/OUT) Must be initialized to -1 and will be returned as -1 when all data has been read.

pData

(OUT) Points to the current entry.

Return Values

0x0000 0000	SUCCESSFUL
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
ERR_ITR_INVALID_SEARCH_DATA	Indicates entry data is in an unexpected format.
ERR_EOF_HIT	Indicates the iterator is positioned at the end of the list
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (-001 to -799).

Remarks

If the returned data does not fit in the application's buffer, the function returns with partial data. You must then call NWDSItrGetCurrent repeatedly with the current value of the iteration handle to get the remaining data. When the iteration handle is returned as -1, all the data has been retrieved.

For sample code, see Section 4.8, "Positioning the Iterator with Typedown: Example," on page 79.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrGetNext (page 48), NWDSItrGetPrev (page 52), NWDSItrSkip (page 58)

NWDSItrGetInfo

Returns information about an iterator object.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrGetInfo (
   nuint_ptr    Iterator,
   pnbool8      pIsScalable,
   pnbool8      pIsPositionable);
```

Parameters

Iterator

(IN) Specifies the iterator object.

pIsScalable

(OUT) Points to a value that specifies whether the iterator object was created on a server running NDS 8 (optional).

N_TRUE	Indicates the iterator is on an NDS 8 server
N_FALSE	Indicates the iterator is running in emulation mode on an NDS/eDirectory server that is not running NDS 8.

pIsPositionable

(OUT) Points to a value that specifies whether the iterator object is positionable (optional).

N_TRUE	Indicates that the positioning functions work with this iterator.
N_FALSE	Indicates that the positioning functions don't work with this iterator.

Return Values

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

Remarks

An N_TRUE value for the pIsPositionable parameter indicates that the NWDSItrGetPosition and NWDSItrSetPosition functions work with this iterator. An iterator created on a eDirectory server not running NDS 8 always returns N_TRUE.

Iterator objects are positionable if they also use a search filter that uses only the same attributes as the index. For example, if an iterator object uses the _BaseClass, _RDN index, the search filter must use only the _BaseClass and the _RDN symbols for attributes for the iterator to be positionable. For example, the following search filters would make the iterator positionable:

"BaseClass=*"

"BaseClass=User"

"RDN=S*"

"BaseClass=User AND RDN=S*"

There are other conditions that can cause an iterator to be nonpositionable. For more information, see “Search Filters” on page 13.

For sample code, see Section 4.5, “Getting Iterator Information: Example,” on page 73.

NCP Calls

None

See Also

NWDSItrCreateList (page 35), NWDSItrCreateSearch (page 38)

NWDSItrGetNext

Returns the number of specified next entries, including the current entry.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrGetNext (
   nuint_ptr    Iterator,
   nuint32      numEntries,
   nuint32      timeout,
   pnint32      pIterationHandle,
   pBuf_T       pData);
```

Parameters

Iterator

(IN) Specifies the iterator object.

numEntries

(IN) Specifies the number of entries to read. Zero indicates that all entries should be read.

timeout

(IN) Specifies the time (in milliseconds) allowed before returning an error.

pIterationHandle

(IN/OUT) Must be initialized to -1 and will be returned as -1 when all entries have been read.

pData

(OUT) Points to the returned entries.

Return Values

0x0000 0000	SUCCESSFUL

ERR_EOF_HIT	Indicates the iterator was already positioned at the EOF and there is no data to read.
ERR_QUERY_TIMEOUT	Indicates the function timed out before retrieving all the requested data.
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
ERR_ITR_INVALID_SEARCH_DATA	Indicates entry data is in an unexpected format.
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (-001 to -799).

Remarks

If the returned data does not fit in the application's buffer, the function returns with partial data. You must then call the NWDSItrGetNext function repeatedly with the current value of the iteration handle to get the remaining data. When the iteration handle is returned as -1, all the data has been retrieved.

This function leaves the current position pointing to the next unread entry after the last one that was retrieved. If the end of the list is reached, the current position is left at EOF.

The return data in the pData parameter is in the same format as the data in the output buffer for the NWDSSearch function. You will need to use the specialized functions such as NWDSGetObjectCount and NWDSGetAttrName to retrieve the data. For step-by-step instructions, see Section 2.2, "Retrieving and Unpacking Object and Attribute Name Data," on page 21 or Section 2.3, "Retrieving and Unpacking Object, Attribute, and Value Data," on page 22.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrGetCurrent (page 44), NWDSItrGetPrev (page 52), NWDSItrSkip (page 58)

NWDSItrGetPosition

Returns the iterator's current logical position.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrGetPosition (
   nuint_ptr    Iterator,
   pnuint32     pPosition,
   nuint32      timeout);
```

Parameters

Iterator

(IN) Specifies the iterator object.

pPosition

(OUT) Returns the current iterator position as a logical position in the 0-1000 range or DS_ITR_EOF (1001). The value is an approximation.

timeout

(IN) Specifies the time (in milliseconds) allowed before returning an error.

Return Values

0x0000 0000	SUCCESSFUL
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
ERR_EOF_HIT	Indicates that current position is beyond the last entry in the list.This is a valid position.
ERR_NOT_IMPLEMENTED	Indicates that this iterator object does not support positioning. For more information, see the NWDSItrGetInfo (page 46) function.

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

Remarks

Unless the iterator object is on one of the special positions, the position returned is an estimate, not an exact location. For more information, see Section 1.4, “Positions of an Iterator Object,” on page 15.

The following values indicate special positions for the pPosition parameter:

0 (DS_ITR_FIRST)	Indicates the current position is at the top of the list
1000 (DS_ITR_LAST)	Indicates the current position is at the last entry in the list
1001 (DS_ITR_EOF)	Indicates the position is at EOF, or just after the last entry in the list

If the position is at EOF, DS_ITR_EOF (1001) is returned for the position and 0 is returned as the status code.

Not all iterator objects are positionable. Such iterator objects return an error code (ERR_NOT_IMPLEMENTED). To determine if the iterator object supports positioning, see the NWDSItrGetInfo (page 46) function.

For sample code, see Section 4.6, “Getting and Setting the Iterator's Position: Example,” on page 74.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrAtEOF (page 26), NWDSItrAtFirst (page 28), NWDSItrSetPosition (page 54), NWDSItrSetPositionFromIterator (page 56)

NWDSItrGetPrev

Returns the number of specified previous entries, not including the current entry.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrGetPrev (
   nuint_ptr   Iterator,
   nuint32     numEntries,
   nuint32     timeout,
   pnint32     pIterationHandle,
   pBuf_T      pData);
```

Parameters

Iterator

(IN) Specifies the iterator object.

numEntries

(IN) Specifies the number of entries to read. Zero indicates all entries should be read.

timeout

(IN) Specifies the time (in milliseconds) allowed before returning an error.

pIterationHandle

(IN/OUT) Must be initialized to -1 and will be returned as -1 when all entries have been read.

pData

(OUT) Points to the returned entries.

Return Values

0x0000 0000	SUCCESSFUL

ERR_BOF_HIT	Indicates the iterator was already positioned at the first entry or the list is empty. There is no data to read.
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
ERR_ITR_INVALID_SEARCH_DATA	Indicates entry data is in an unexpected format.
ERR_QUERY_TIMEOUT	Indicates the function timed out before it retrieved all the requested data.
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (-001 to -799).

Remarks

The entries are returned in REVERSE sort order. The current position is left pointing to the last entry returned (the one closest to the top of the list). If an attempt is made to read beyond the top of the list, the current position is left at the first entry.

If the returned data does not fit in the result buffer, the function returns with partial information. If the pIterationHandle is not returned as -1, all the data has not been retrieved. You must then call NWDSItrGetPrev again with the current value of the iteration handle to get the remaining data. When the iteration handle is returned as -1, all the data has been retrieved.

The returned data in the pData parameter is in the same format as NWDSSearch, and must be unpacked with the same specialized functions. All data must be unpacked; there is no way to skip data. For step-by-step instructions, see Section 2.2, "Retrieving and Unpacking Object and Attribute Name Data," on page 21 and Section 2.3, "Retrieving and Unpacking Object, Attribute, and Value Data," on page 22.

For sample code, see Section 4.7, "Listing in Reverse Order: Example," on page 77.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrGetCurrent (page 44), NWDSItrGetNext (page 48), NWDSItrSkip (page 58)

NWDSItrSetPosition

Sets the iterator's current position according to a logical value.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrSetPosition (
   nuint_ptr   Iterator,
   nuint32     position,
   nuint32     timeout);
```

Parameters

Iterator

(IN) Specifies the iterator object.

position

(IN) Specifies the position to set the iterator object.

timeout

(IN) Specifies the time (in milliseconds) allowed before returning an error.

Return Values

0x0000 0000	SUCCESSFUL
ERR_QUERY_TIMEOUT	Indicates the function reached its timeout value before it could successfully position the iterator object.
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
ERR_ITR_INVALID_POSITION	Indicates that the logical position specified was not in the 0 to 1000 range.
ERR_NOT_IMPLEMENTED	Indicates that the iterator object does not support positioning. For more information, see the NWDSItrGetInfo (page 46) function.

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

Remarks

The position parameter takes an integer value from 0 to 1000. DS_ITR_FIRST(0) and DS_ITR_LAST(1000) will set the position to the first or last entry. DS_ITR_EOF (1001) will set the position to EOF (End-Of-File), just after the last entry. All iterator objects allow these special positions to be set.

Except for the special positions, the position set is an approximation. A subsequent call to get the current position will probably return a different value.

Not all iterator objects are positionable. If the iterator is not positionable, the function returns an error code (ERR_NOT_IMPLEMENTED). To determine if the iterator supports positioning, see the NWDSItrGetInfo (page 46) function.

For sample code, see Section 4.6, “Getting and Setting the Iterator's Position: Example,” on page 74.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrGetPosition (page 50), NWDSItrSetPositionFromIterator (page 56), NWDSItrTypeDown (page 60), NWDSItrGetInfo (page 46)

NWDSItrSetPositionFromIterator

Sets the iterator's current position to the source iterator's position.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrSetPositionFromIterator (
   nuint_ptr   Iterator,
   nuint_ptr   srcIterator,
   nuint32      timeout);
```

Parameters

Iterator

(IN) Specifies the iterator object that will have its position changed.

srcIterator

(IN) Specifies the source iterator.

timeout

(IN) Specifies the time (in milliseconds) allowed before returning an error.

Return Values

0x0000 0000	SUCCESSFUL
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object or that one iterator was from an NDS 8 server and the other one was not.
nonzero value	Nonzero values indicate errors. See "NDS Return Values" (-001 to -799).

Remarks

For NDS 8, the two iterators do not need to be identical. The system tries to find the closest match in the srcIterator and positions the destination one accordingly.

For NDS server not running NDS 8, the Iterator must be a clone of the srcIterator.

Both iterators must either be from NDS 8 servers or from servers not running NDS 8. If one references an NDS 8 server and the other a server that is not running NDS 8, an error (ERR_ITR_INVALID_HANDLE) is returned.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrGetPosition (page 50), NWDSItrSetPosition (page 54), NWDSItrTypeDown (page 60)

NWDSItrSkip

Skips entries, either forward or backward.

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrSkip (
   nuint_ptr     Iterator,
   nint32        numToSkip,
   nuint32       timeout,
   pnint32       pNumSkipped);
```

Parameters

Iterator

(IN) Specifies the iterator object.

numToSkip

(IN) Specifies the number of entries to skip. May be positive (forward in the list) or negative (backward in the list).

timeout

(IN) Specifies the time (in milliseconds) allowed before returning an error.

pNumSkipped

(OUT) Points to the number of entries actually skipped. May be NULL. The number will be positive for a forward skip or negative for a backward skip.

Return Values

0x0000 0000	SUCCESSFUL
ERR_BOF_HIT	Indicates that the iterator was already positioned at the top of the list. No entries were skipped.

ERR_EOF_HIT	Indicates that the iterator was already positioned at the EOF. No entries were skipped.
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (-001 to -799).

Remarks

You can request to skip to a position that is beyond the beginning or the end of the list.

- If you are skipping back to the beginning of the list, the function positions the iterator at the first entry in the list and places the actual number of entries in the pNumSkipped parameter. If the iterator is already positioned on the first entry, the function returns ERR_BOF_HIT, and the iterator remains on the first entry.
- If you are skipping forward to the end of the list and the request takes you beyond the end, the function places the iterator one entry beyond the end of the list and places the actual number of entries in the pNumSkipped parameter. If the iterator is already positioned at EOF, the function returns ERR_EOF_HIT, and the iterator remains in the EOF position.

A skip with numToSkip set to zero always returns success with no change in position.

For sample code, see Section 4.9, “Skipping Objects in the List: Example,” on page 81.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrGetCurrent (page 44), NWDSItrGetNext (page 48), NWDSItrGetPrev (page 52)

NWDSItrTypeDown

Sets the iterator position according to the specified attribute and value and implements “typedown.”

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: NDS Iterator

Syntax

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

NWDSCCODE  NWDSItrTypeDown (
   nuint_ptr   Iterator,
   pnstr8      attrString,
   pnstr8      value,
   nuint32     byteUniFlag,
   nuint32     timeout);
```

Parameters

Iterator

(IN) Specifies the iterator object.

attrString

(IN) Specifies the attribute to use for typedown. It is usually set to NULL to select the default attribute for the index of the iterator object.

value

(IN) Specifies the string value to use for typedown positioning.

byteUniFlag

(IN) Specifies whether the input is a byte or Unicode string:

0	DS_ITR_UNICODE_STRING: Indicates a Unicode string.
2	DS_ITR_BYTE_STRING: Indicates a byte string.

timeout

(IN) Specifies the time (in milliseconds) allowed before returning an error.

Return Values

0x0000 0000	SUCCESSFUL
ERR_ITR_INVALID_HANDLE	Indicates the Iterator parameter was not a valid iterator object.
nonzero value	Nonzero values indicate errors. See “NDS Return Values” (-001 to -799).

Remarks

If you have a list sorted by surname, you can specify “D” and it will position to the first name starting with D. If you specify “DA”, it positions to the first name starting with DA, etc. If there are no entries matching the value string, it positions to the first one that is greater than the specified value. If no entries have a greater value, the position is set to EOF.

If you have set context handle flags with the NWDSSetContext function, the byteUniFlag parameter can use the flags parameter from the NWDSSetContext function as its value.

This function can give unexpected results when used with the _BaseClass,_RDN index if the search filter allows for multiple base classes. The index sorts objects first by base class and then alphabetically within the class. For example, an index of the following groups and users would produce unexpected results.

```
Groups
   Cat Lovers
   Dog Lovers
   Horse Lovers

Users
  Ann
  Chris
  Don
  Kim
  Zed
```

A typedown of "Chr" would position the iterator on "Dog Lovers," the first object it finds with a value greater than "Cat Lovers" object. However, a typedown of "Ki" would position the iterator on Kim. If the Group base class had an entry of "Zebra Lovers," only users with names that started with a value greater than "Zebra Lovers" (such as Zed) could be returned on a typedown.

For sample code, see Section 4.8, “Positioning the Iterator with Typedown: Example,” on page 79.

NCP Calls

0x2222 104 02 Send eDirectory Fragmented Request/Reply

See Also

NWDSItrSetPosition (page 54), NWDSItrSetPositionFromIterator (page 56), NWDSItrGetPosition (page 50)

eDirectory Iterator Example Code

4

This chapter provides example code for the common tasks when using Novell® eDirectory™ Iterator services.

4.1 Cloning an Iterator Object: Example

```
/* itrclone.c
   Example of NDS 8 Iterator clone function.
*/

#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwcalls.h>
#include <assert.h>

void main(int argc, char *argv[])
{
   NWDSContextHandle context =
         (NWDSContextHandle)ERR_CONTEXT_CREATION;
   nstr8          strAbbreviatedName[MAX_DN_CHARS+1];
   pBuf_T         pb = NULL;
   NWDSCCODE      ccode;
   nuint32        flags;
   nuint32        origIter=0;
   nuint32        cloneIter=0;
   nuint32        timeout = 0;     /* Timeout=0 means no time limit. */

/* Check arguments */
   if(argc < 2)
   {
      printf("\nUsage:    itrclone  <base object>\n");
      printf("\nExample:  itrclone  myou.myorg\n");
      exit(1);
   }
```

```
/* Initialize NWCalls and unicode tables */
ccode = NWCallsInit(NULL,NULL);
   if(ccode) exit(1);

/* Create NDS Context */
ccode = NWDSCreateContextHandle(&context);
if(ccode) exit(1);

/* Set flags to get object names in typeless format. */
NWDSGetContext(context, DCK_FLAGS, &flags);
flags |= DCV_TYPELESS_NAMES;
NWDSSetContext(context, DCK_FLAGS, &flags);

printf("itrclone:   Sample program for Iterator informational
        functions.\n");
printf("Container:    %s\n", argv[1]);

/* Convert the directory name (passed in as first arg) to its
   shortest form relative to the name context */
ccode = NWDSAbbreviateName(context, argv[1], strAbbreviatedName);
if(ccode) goto error;

/* Allocate the output buffer.  */
ccode = NWDSAllocBuf(MAX_MESSAGE_LEN, &pb);
if (ccode) goto error;

/* ———— Create the Iterator ————- */

ccode = NWDSItrCreateList(
              context,              /* context for this search */
              strAbbreviatedName,   /* container to search     */
              NULL,                 /* class name filter       */
              NULL,                 /* subordinate name filter */
              DS_ITR_ANY_SERVER,    /* scalability requirement */
              timeout,              /* 0 = no timeout          */
              &origIter);           /* returned Iterator ptr   */
if (ccode) goto error;

/* Create a clone of the original iterator */
ccode = NWDSItrClone(origIter, &cloneIter);
if (ccode) goto error;

/* Position the clone to EOF */
ccode = NWDSItrSetPosition(cloneIter, DS_ITR_EOF, timeout);
if (ccode) goto error;

/* The original iter should still be at the first position. */
/* The clone iterator should be at EOF. */
assert(NWDSItrAtFirst(origIter) == N_TRUE);
assert(NWDSItrAtEOF(cloneIter) == N_TRUE);

/* Position the clone iterator to the same position as the
   original. */
ccode = NWDSItrSetPositionFromIterator(cloneIter, origIter,
```

```
            timeout);
   if (ccode) goto error;
   assert(NWDSItrAtFirst(cloneIter) == N_TRUE);

/* Clean up, normal or error termination. */
error:
   printf("\n\nccode = %d.",ccode);
   if (origIter)
     NWDSItrDestroy(origIter);   /* Destroy the Iterator when done. */
   if (cloneIter)
      NWDSItrDestroy(cloneIter);  /* Destroy the clone also. */
   if (pb)
      NWDSFreeBuf(pb);
   NWDSFreeContext(context);
}
```

4.2 Counting with eDirectory Iterators: Example

```
/* itrcount.c
   Example of NDS 8 Iterator NWDSItrCount function.
*/

#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwcalls.h>

void main(int argc, char *argv[])
{
   NWDSContextHandle context =
      (NWDSContextHandle)ERR_CONTEXT_CREATION;
   nstr8        strAbbreviatedName[MAX_DN_CHARS+1];
   NWDSCCODE    ccode;
   nuint32      flags;
   nuint32      iter=0;
   nint32       iterHandle = -1;
   nuint32      timeout = 0;    /* Timeout=0 means no time limit. */
   nuint32      limit= 0;
   nuint32      count;

/* Check arguments */
   if(argc < 2)
   {
      printf("\nUsage:    itrcount  <base object>  [count limit]\n");
      printf("\nExample:  itrcount  myou.myorg  1000\n");
      exit(1);
   }

   if (argc >= 3)
      sscanf(argv[2], "%d", &limit);

   /* Initialize NWCalls and unicode tables */
```

```
   ccode = NWCallsInit(NULL,NULL);
      if(ccode) exit(1);

   /* Create NDS Context */
   ccode = NWDSCreateContextHandle(&context);
   if(ccode) exit(1);

   /* Set flags to get object names in typeless format. */
   NWDSGetContext(context, DCK_FLAGS, &flags);
   flags |= DCV_TYPELESS_NAMES;
   NWDSSetContext(context, DCK_FLAGS, &flags);

   printf("itrcount:   Sample program for NWDSItrCount\n");
   printf("Container:    %s\n", argv[1]);
   printf("Count limit:  %d\n\n", limit);

   /* Convert the directory name (passed in as first arg) to its
      shortest form relative to the name context */
   ccode = NWDSAbbreviateName(context, argv[1], strAbbreviatedName);
   if(ccode) goto error;

   /* ——— Create the Iterator ———- */

   ccode = NWDSItrCreateList(
                context,              /* context for this search */
                strAbbreviatedName,   /* container to search     */
                NULL,                 /* class name filter       */
                NULL,                 /* subordinate name filter */
                DS_ITR_ANY_SERVER,    /* scalability requirement */
                timeout,              /* 0 = no timeout          */
                &iter);               /* returned Iterator ptr   */
   if (ccode)
      goto error;


   /* Count until we reach the max count limit, timeout, or end
      of list. Do not update the current position while counting.
   */
   ccode = NWDSItrCount(iter, timeout, limit, N_FALSE, &count);
   if (ccode == ERR_QUERY_TIMEOUT)
      printf("Count timed out after counting %d objects\n",count);
   else if (ccode == ERR_ITR_MAX_COUNT)
      printf("Count reached max count limit: %d\n", count);
   else
      printf("Count reached the end of the list.  %d
              objects.\n",count);


/* Clean up, normal or error termination. */
error:
   printf("\n\nccode = %d.",ccode);
   if (iter)
     NWDSItrDestroy(iter);      /* Destroy the Iterator when done. */
```

```
   NWDSFreeContext(context);
}
```

4.3 Creating and Using a List Iterator: Example

```
/* itrlist.c
   Example of creating a list iterator of selected class and object
   names, and printing the object names.
*/

#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwcalls.h>

void main(int argc, char *argv[])
{
   NWDSContextHandle context =
         (NWDSContextHandle)ERR_CONTEXT_CREATION;
   nstr8             strAbbreviatedName[MAX_DN_CHARS+1];
   pBuf_T            pb = NULL;
   NWDSCCODE         ccode;
   nuint32           flags;
   nuint32           iter=0;
   nint32            iterHandle = -1;

/* Check arguments */
   if(argc != 4)
   {
      printf("\nUsage:    itrlist  <base object>  <object class>
              <name filter>");
      printf("\nExample:  itrlist  myou.myorg  user a*");
      exit(1);
   }

   /* Initialize NWCalls and unicode tables */
   ccode = NWCallsInit(NULL,NULL);
      if(ccode) exit(1);

   /* Create NDS Context */
   ccode = NWDSCreateContextHandle(&context);
   if(ccode) exit(1);

   /* Set flags to get object names in typeless format. */
   NWDSGetContext(context, DCK_FLAGS, &flags);
   flags |= DCV_TYPELESS_NAMES;
   NWDSSetContext(context, DCK_FLAGS, &flags);

   printf("itrlist:   Sample program for NDS 8 Iterator List\n");
   printf("Container to list:    %s\n",   argv[1]);
   printf("Class:                %s\n",   argv[2]);
   printf("Name filter:          %s\n",   argv[3]);
```

```
/* Convert the directory name (passed in as first arg) to its
   shortest form relative to the name context */
ccode = NWDSAbbreviateName(context, argv[1], strAbbreviatedName);
if(ccode) goto error;

/* Allocate the output buffer.  */
ccode = NWDSAllocBuf(MAX_MESSAGE_LEN, &pb);
if(ccode) goto error;

/* ——— Create the Iterator ———-
   A list iterator is ordered first by Base Class,
   then by RDN within a base class.
*/

ccode = NWDSItrCreateList(
             context,              /* context for this search */
             strAbbreviatedName,   /* container to search     */
             argv[2],              /* class name filter       */
             argv[3],              /* subordinate name filter */
             DS_ITR_ANY_SERVER,    /* scalability requirement */
             0,                    /* no timeout              */
             &iter);               /* returned Iterator ptr   */
if (ccode)
   goto error;

/*
   Read all entries in the result set and dump the results.
   If the search result is empty, -765 (ERR_EOF_HIT) is returned.
*/
do              /* Loop until Iteration handle returns -1 */
{
   nuint      i;
   nstr8      objName[MAX_DN_CHARS+1];
   nuint32    objCount, attrCount;
   Object_Info_T  objectInfo;

   ccode = NWDSItrGetNext(
                iter,          /* Iterator object handle      */
                0,             /* # of entries to read. 0=all */
                0,             /* no timeout                  */
                &iterHandle,   /* Iteration handle            */
                pb);           /* Returned data               */
   if (ccode) goto error;

   ccode = NWDSGetObjectCount (context,pb,&objCount);
   if (ccode) goto error;
   printf("\nNext buffer.  Object Count = %d\n", objCount);

   for (i=0; i<objCount; i++)
   {
      ccode = NWDSGetObjectName (context,
                                 pb,
                                 objName,
                                 &attrCount,
```

```
                                    &objectInfo);
         if (ccode) goto error;
         printf("  %-20s     Class: %s\n",
                objName,objectInfo.baseClass);

      }  /* For each object */
   } while (iterHandle != -1);

/* Clean up, normal or error termination. */
error:
   printf("\n\nccode = %d.",ccode);
   if (iter)
      NWDSItrDestroy(iter);    /* Destroy the Iterator when done. */
   if (pb)
      NWDSFreeBuf(pb);
   NWDSFreeContext(context);
}
```

4.4 Creating a Search Iterator and Displaying the Results: Example

```
/* itrsrch.c
   Example of creating a search iterator and displaying all text
   attributes.
*/

#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwcalls.h>

void main(int argc, char *argv[])
{
   NWDSContextHandle context =
          (NWDSContextHandle)ERR_CONTEXT_CREATION;
   nstr8             strAbbreviatedName[MAX_DN_CHARS+1];
   pBuf_T            pFilter = NULL;
   pBuf_T            pb = NULL;
   pFilter_Cursor_T  pCur = NULL;
   NWDSCCODE         ccode;
   nuint32           flags;
   nuint32           iter=0;
   nint32            iterHandle = -1;

/* Check arguments */
   if(argc != 3)
   {
      printf("\nUsage:    itrsrch  <base object>  <object class>");
      printf("\nExample:  itrsrch  myou.myorg  user");
      exit(1);
   }

   /* Initialize NWCalls and unicode tables */
```

```
ccode = NWCallsInit(NULL,NULL);
   if(ccode) exit(1);

/* Create NDS Context */
ccode = NWDSCreateContextHandle(&context);
if(ccode) exit(1);

/* Set flags to get object names in typeless format. */
NWDSGetContext(context, DCK_FLAGS, &flags);
flags |= DCV_TYPELESS_NAMES;
NWDSSetContext(context, DCK_FLAGS, &flags);

printf("itrsrch:   Sample program for NDS 8 Iterator Search\n");
printf("Container to search:  %s\n",   argv[1]);
printf("Class:                %s\n",   argv[2]);

/* Convert the directory name (passed in as first arg) to its
   shortest form relative to the name context */
ccode = NWDSAbbreviateName(context, argv[1], strAbbreviatedName);
if(ccode) goto error;

/* Allocate the output buffer.  */
ccode = NWDSAllocBuf(MAX_MESSAGE_LEN, &pb);
if(ccode) goto error;

/* Allocate space for the search filter cursor. */
ccode = NWDSAllocFilter(&pCur);
if(ccode) goto error;

/* Allocate the buffer to hold the search filter.  */
ccode = NWDSAllocBuf(DEFAULT_MESSAGE_LEN, &pFilter);
if(ccode) goto error;

/* Initialize filter for a DSV_SEARCH_FILTER operation */
ccode = NWDSInitBuf(context, DSV_SEARCH_FILTER, pFilter);
if(ccode) goto error;

/* Filter expression "FTOK_BASECLS OBJCLASS". */
ccode = NWDSAddFilterToken(pCur, FTOK_BASECLS, NULL,0);
if(ccode) goto error;

ccode = NWDSAddFilterToken(pCur, FTOK_ANAME, argv[2],
        SYN_CLASS_NAME);
if(ccode) goto error;

ccode = NWDSAddFilterToken(pCur, FTOK_END, NULL, 0);
if(ccode) goto error;

/* Place finished search filter into the search input buffer */
ccode = NWDSPutFilter(context, pFilter, pCur, NULL);
pCur = NULL;  /* NWDSPutFilter frees the expression tree memory. */
if(ccode) goto error;
```

```
/* ——— Create the Iterator ———-
   Since indexSelect is NULL, the server chooses the optimum
   index for this query.  The index chosen will determine the
   sort order of the result set.  You may also explicitly specify
   which index to use.
*/

ccode = NWDSItrCreateSearch(
              context,              /* context for this search */
              strAbbreviatedName,   /* container to search     */
              DS_SEARCH_SUBORDINATES, /* scope                 */
              N_FALSE,              /* deref alias false       */
              pFilter,              /* search filter           */
              NULL,                 /* Time Stamp filter       */
              DS_ATTRIBUTE_VALUES,  /* info type to return     */
              N_TRUE,               /* return all attributes   */
              NULL,                 /* attribute list to ret   */
              NULL,                 /* index select            */
              NULL,                 /* emulation mode sort key */
              DS_ITR_ANY_SERVER,    /* scalability requirement */
              0,                    /* no timeout              */
              &iter);               /* returned Iterator ptr   */
if (ccode)
   goto error;

/*
   Read all entries in the result set and dump the results.
   If the search result is empty, -765 (ERR_EOF_HIT) is returned.
*/
do                /* Loop until Iteration handle returns -1 */
{
   nuint      i,j,k;
   nstr8      objName[MAX_DN_CHARS+1];
   nstr8      attrName[MAX_DN_CHARS+1];
   nuint8     attrVal[MAX_MESSAGE_LEN];
   nuint32    objCount, attrCount, attrValCount, syntaxID;
   Object_Info_T  objectInfo;

   ccode = NWDSItrGetNext(
                    iter,        /* Iterator object handle     */
                    0,           /* # of entries to read. 0=all */
                    0,           /* no timeout                 */
                    &iterHandle, /* Iteration handle           */
                    pb);         /* Returned data              */
   if (ccode) goto error;

   ccode = NWDSGetObjectCount (context,pb,&objCount);
   if (ccode) goto error;
   printf("\nNext buffer.  Object Count = %d\n", objCount);

   for (i=0; i<objCount; i++)
   {
      ccode = NWDSGetObjectName (context,
                                 pb,
```

```
                                    objName,
                                    &attrCount,
                                    &objectInfo);
          if (ccode) goto error;
         printf("\n %s    Class: %s\n", objName,objectInfo.baseClass);

          for (j=0; j<attrCount; j++)  /* If we returned attrs,
                                          print them. */
          {
             ccode =  NWDSGetAttrName (context,
                                     pb,
                                     attrName,
                                     &attrValCount,
                                     &syntaxID);
             printf("    Attr: %s   syntax=%d   ValCount=%d\n",
                    attrName,syntaxID,attrValCount);
             if (ccode) goto error;

             for (k=0; k<attrValCount; k++)
             {
                ccode =  NWDSGetAttrVal(context,
                                     pb,
                                     syntaxID,
                                     attrVal);
                if (ccode) goto error;

                /* If this is a text attribute, print it. */
               if ((syntaxID >= 1 && syntaxID <= 5) || syntaxID == 20)
                    printf("      Value:  %s\n",attrVal);
             }  /* For each value */
          }  /* For each attr */
       }  /* For each object */
    } while (iterHandle != -1);

/* Clean up, normal or error termination. */
error:
    printf("\n\nccode = %d.",ccode);
    if (iter)
       NWDSItrDestroy(iter);     /* Destroy the Iterator when done. */
    if (pb)
       NWDSFreeBuf(pb);
    if (pCur)
       NWDSFreeFilter(pCur,NULL);
    if (pFilter)
       NWDSFreeBuf(pFilter);
    NWDSFreeContext(context);
}
```

4.5 Getting Iterator Information: Example

```
/* itrinfo.c
   Example of NDS 8 Iterator informational functions:
      NWDSItrGetInfo
      NWDSItrAtFirst
      NWDSItrAtEOF
*/

#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwcalls.h>

void main(int argc, char *argv[])

{
   NWDSContextHandle context =
         (NWDSContextHandle)ERR_CONTEXT_CREATION;
   nstr8        strAbbreviatedName[MAX_DN_CHARS+1];
   NWDSCCODE    ccode;
   nuint32      flags;
   nuint32      iter=0;
   nuint32      timeout = 0;     /* Timeout=0 means no time limit. */
   nbool8       isScalable;
   nbool8       isPositionable;

/* Check arguments */
   if(argc < 2)
   {
      printf("\nUsage:    itrinfo  <base object>\n");
      printf("\nExample:  itrinfo  myou.myorg\n");
      exit(1);
   }

   /* Initialize NWCalls and unicode tables */
   ccode = NWCallsInit(NULL,NULL);
      if(ccode) exit(1);

   /* Create NDS Context */
   ccode = NWDSCreateContextHandle(&context);
   if(ccode) exit(1);

   /* Set flags to get object names in typeless format. */
   NWDSGetContext(context, DCK_FLAGS, &flags);
   flags |= DCV_TYPELESS_NAMES;
   NWDSSetContext(context, DCK_FLAGS, &flags);

   printf("itrinfo:   Sample program for Iterator informational
           functions.\n");
   printf("Container:    %s\n", argv[1]);

   /* Convert the directory name (passed in as first arg) to its
```

```
      shortest form relative to the name context */
   ccode = NWDSAbbreviateName(context, argv[1], strAbbreviatedName);
   if(ccode) goto error;

   /* ——— Create the Iterator ———- */

   ccode = NWDSItrCreateList(
               context,              /* context for this search */
               strAbbreviatedName,   /* container to search     */
               NULL,                 /* class name filter       */
               NULL,                 /* subordinate name filter */
               DS_ITR_ANY_SERVER,    /* scalability requirement */
               timeout,              /* 0 = no timeout          */
               &iter);               /* returned Iterator ptr   */
   if (ccode)
      goto error;


   ccode = NWDSItrGetInfo(iter, &isScalable, &isPositionable);
   if (ccode) goto error;

   if (isScalable)
      printf("  This Iterator is SCALABLE.\n");
   else
      printf("  This Iterator is NOT SCALABLE.\n");

   if (isPositionable)
      printf("  This Iterator is POSITIONABLE.\n");
   else
      printf("  This Iterator is NOT POSITIONABLE.\n");

   if (NWDSItrAtFirst(iter))
      printf("  Iterator is positioned at first entry.\n");

   if (NWDSItrAtEOF(iter))
      printf("  Iterator is positioned at EOF, or is empty.\n");


/* Clean up, normal or error termination. */
error:
   printf("\n\nccode = %d.",ccode);
   if (iter)
      NWDSItrDestroy(iter);      /* Destroy the Iterator when done. */
   NWDSFreeContext(context);
}
```

4.6 Getting and Setting the Iterator's Position: Example

```
/* itrpos.c
   Example of NDS 8 Iterator GetPosition and SetPosition calls.
*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwcalls.h>

void main(int argc, char *argv[])
{
   NWDSContextHandle context =
          (NWDSContextHandle)ERR_CONTEXT_CREATION;
   nstr8          strAbbreviatedName[MAX_DN_CHARS+1];
   NWDSCCODE      ccode;
   nuint32        flags;
   nuint32        iter=0;
   nint32         iterHandle = -1;
   nuint32        timeout = 0;     /* Timeout=0 means no time limit. */
   nuint32        pos;

/* Check arguments */
   if(argc != 2)
   {
      printf("\nUsage:    itrpos  <base object>");
      printf("\nExample:  itrpos  myou.myorg");
      exit(1);
   }

   /* Initialize NWCalls and unicode tables */
   ccode = NWCallsInit(NULL,NULL);
      if(ccode) exit(1);

   /* Create NDS Context */
   ccode = NWDSCreateContextHandle(&context);
   if(ccode) exit(1);

   /* Set flags to get object names in typeless format. */
   NWDSGetContext(context, DCK_FLAGS, &flags);
   flags |= DCV_TYPELESS_NAMES;
   NWDSSetContext(context, DCK_FLAGS, &flags);

   printf("itrpos:   Sample program for NDS 8 NWDSItrGet/
           SetPosition\n");
   printf("Container:    %s\n",   argv[1]);

   /* Convert the directory name (passed in as first arg) to its
      shortest form relative to the name context */
   ccode = NWDSAbbreviateName(context, argv[1], strAbbreviatedName);
   if(ccode) goto error;

   /* ——— Create the Iterator ———- */

   ccode = NWDSItrCreateList(
                context,              /* context for this search */
                strAbbreviatedName,   /* container to search     */
                NULL,                 /* class name filter       */
```

```
                NULL,                   /* subordinate name filter */
                DS_ITR_ANY_SERVER,      /* scalability requirement */
                timeout,                /* 0 = no timeout          */
                &iter);                 /* returned Iterator ptr   */
if (ccode)
   goto error;

/* Do several Get and SetPosition calls.  Iterator positions are
   always "logical positions" in the range 0-1001.  Special
   symbols defined:
      DS_ITR_FIRST = 0
      DS_ITR_LAST  = 1000
      DS_ITR_EOF   = 1001
*/

/* Initial position should be 0.  If list is empty,
   position will be 1001 (EOF). */
printf("\nInitial position is:  ");
ccode = NWDSItrGetPosition(iter, &pos, timeout);
if (ccode == ERR_NOT_IMPLEMENTED)
   printf(" *** Iterator not positionable.\n");
else if (ccode)
   goto error;
else
   printf("%d\n",pos);

printf("Setting position 500.\n");
ccode = NWDSItrSetPosition(iter, 500, timeout);
if (ccode == ERR_NOT_IMPLEMENTED)
   printf("  *** Iterator not positionable.\n");
else if (ccode)
   goto error;

printf("Current position is now:  ");
ccode = NWDSItrGetPosition(iter, &pos, timeout);
if (ccode == ERR_NOT_IMPLEMENTED)
   printf(" *** Iterator not positionable.\n");
else if (ccode)
   goto error;
else
   printf("%d\n",pos);

/* You should always be able to set positions 0, 1000, and 1001,
   even if the iterator is not positionable. */
printf("Setting position 1000.\n");
ccode = NWDSItrSetPosition(iter, DS_ITR_LAST, timeout);
if (ccode) goto error;

printf("Current position is now:  ");
ccode = NWDSItrGetPosition(iter, &pos, timeout);
if (ccode == ERR_NOT_IMPLEMENTED)
   printf(" *** Iterator not positionable.\n");
else if (ccode)
   goto error;
```

```
   else
      printf("%d\n",pos);


/* Clean up, normal or error termination. */
error:
   printf("\n\nccode = %d.",ccode);
   if (iter)
      NWDSItrDestroy(iter);      /* Destroy the Iterator when done. */
   NWDSFreeContext(context);
}
```

4.7 Listing in Reverse Order: Example

```
/* itrprev.c
   Example of processing a list in reverse order using NWDSItrPrev.
*/

#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwcalls.h>

void main(int argc, char *argv[])
{
   NWDSContextHandle context =
        (NWDSContextHandle)ERR_CONTEXT_CREATION;
   nstr8             strAbbreviatedName[MAX_DN_CHARS+1];
   pBuf_T            pb = NULL;
   NWDSCCODE         ccode;
   nuint32           flags;
   nuint32           iter=0;
   nint32            iterHandle = -1;

/* Check arguments */
   if(argc != 3)
   {
      printf("\nUsage:    itrprev  <base object>  <object class>");
      printf("\nExample:  itrprev  myou.myorg  user");
      exit(1);
   }

   /* Initialize NWCalls and unicode tables */
   ccode = NWCallsInit(NULL,NULL);
      if(ccode) exit(1);

   /* Create NDS Context */
   ccode = NWDSCreateContextHandle(&context);
   if(ccode) exit(1);

   /* Set flags to get object names in typeless format. */
```

```
NWDSGetContext(context, DCK_FLAGS, &flags);
flags |= DCV_TYPELESS_NAMES;
NWDSSetContext(context, DCK_FLAGS, &flags);

printf("itrprev:  Sample program for NDS 8 Iterator. List in
        reverse.\n");
printf("Container to list:    %s\n",   argv[1]);
printf("Class:                %s\n",   argv[2]);

/* Convert the directory name (passed in as first arg) to its
   shortest form relative to the name context */
ccode = NWDSAbbreviateName(context, argv[1], strAbbreviatedName);
if(ccode) goto error;

/* Allocate the output buffer.  */
ccode = NWDSAllocBuf(MAX_MESSAGE_LEN, &pb);
if(ccode) goto error;

/* ——— Create the Iterator ———-
   A list iterator is ordered first by Base Class,
   then by RDN within a base class.
*/

ccode = NWDSItrCreateList(
              context,              /* context for this search */
              strAbbreviatedName,   /* container to search     */
              argv[2],              /* class name filter       */
              NULL,                 /* subordinate name filter */
              DS_ITR_ANY_SERVER,    /* scalability requirement */
              0,                    /* no timeout              */
              &iter);               /* returned Iterator ptr   */
if (ccode)
   goto error;

/* Position the Iterator at the EOF position. */
ccode = NWDSItrSetPosition(iter, DS_ITR_EOF, 0);
if (ccode) goto error;

/*
   Read all entries in the result set and dump the results.
   If the search result is empty, -765 (ERR_EOF_HIT) is returned.
*/
do                /* Loop until Iteration handle returns -1 */
{
   nuint          i;
   nstr8          objName[MAX_DN_CHARS+1];
   nuint32        objCount, attrCount;
   Object_Info_T  objectInfo;

   ccode = NWDSItrGetPrev(
                  iter,         /* Iterator object handle      */
                   0,           /* # of entries to read. 0=all */
                   0,           /* no timeout                  */
                   &iterHandle, /* Iteration handle            */
```

```
                        pb);          /* Returned data               */
      if (ccode) goto error;

      ccode = NWDSGetObjectCount (context,pb,&objCount);
      if (ccode) goto error;
      printf("\nNext buffer.  Object Count = %d\n", objCount);

      for (i=0; i<objCount; i++)
      {
         ccode = NWDSGetObjectName (context,
                                    pb,
                                    objName,
                                    &attrCount,
                                    &objectInfo);
         if (ccode) goto error;
         printf("  %-20s     Class: %s\n",
                 objName,objectInfo.baseClass);

      }  /* For each object */
   } while (iterHandle != -1);

/* Clean up, normal or error termination. */
error:
   printf("\n\nccode = %d.",ccode);
   if (iter)
      NWDSItrDestroy(iter);      /* Destroy the Iterator when done. */
   if (pb)
      NWDSFreeBuf(pb);
   NWDSFreeContext(context);
}
```

4.8 Positioning the Iterator with Typedown: Example

```
/* itrtypdn.c
   Example of NDS 8 Iterator Typedown and GetCurrent functions. */

#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwcalls.h>

#define BASECLASS  "User"

void main(int argc, char *argv[])
{
   NWDSContextHandle context =
         (NWDSContextHandle)ERR_CONTEXT_CREATION;
   nstr8          strAbbreviatedName[MAX_DN_CHARS+1];
   pBuf_T         pb = NULL;
   NWDSCCODE      ccode;
   nuint32        flags;
```

```
   nuint32        iter=0;
   nint32         iterHandle = -1;
   nstr8          objName[MAX_DN_CHARS+1];
   nuint32        objCount, attrCount;
   Object_Info_T  objectInfo;

/* Check arguments */
   if(argc != 3)
   {
     printf("\nUsage:    itrtypdn  <base object>  <typedown string>");
      printf("\nExample:  itrtypdn  myou.myorg  d");
      exit(1);
   }

   /* Initialize NWCalls and unicode tables */
   ccode = NWCallsInit(NULL,NULL);
      if(ccode) exit(1);

   /* Create NDS Context */
   ccode = NWDSCreateContextHandle(&context);
   if(ccode) exit(1);

   /* Set flags to get object names in typeless format. */
   NWDSGetContext(context, DCK_FLAGS, &flags);
   flags |= DCV_TYPELESS_NAMES;
   NWDSSetContext(context, DCK_FLAGS, &flags);

   printf("itrtypdn:   Sample program for NDS 8 Iterator List\n");
   printf("Container to list:    %s\n",   argv[1]);
   printf("Typedown string:      %s\n",   argv[2]);

   /* Convert the directory name (passed in as first arg) to its
      shortest form relative to the name context */
   ccode = NWDSAbbreviateName(context, argv[1], strAbbreviatedName);
   if(ccode) goto error;

   /* Allocate the output buffer.  */
   ccode = NWDSAllocBuf(MAX_MESSAGE_LEN, &pb);
   if(ccode) goto error;

   /* ———— Create the Iterator ————-
      A list iterator is ordered first by Base Class,
      then by RDN within a base class.
   */

   ccode = NWDSItrCreateList(
                context,              /* context for this search */
                strAbbreviatedName,   /* container to search     */
                BASECLASS,            /* class name filter       */
                NULL,                 /* subordinate name filter */
                DS_ITR_ANY_SERVER,    /* scalability requirement */
                0,                    /* no timeout              */
                &iter);               /* returned Iterator ptr   */
   if (ccode)
```

```
      goto error;


   /* For a List Iterator the BaseClass,RDN index is used.  Typedown
      on this index uses the RDN field.  The iterator should thus be
      restricted to a single baseclass.
   */
   ccode = NWDSItrTypeDown(
                     iter,     /* Iterator Ptr */
                     NULL,     /* Reserved */
                     argv[2],  /* Typedown string. RDN value */
                     flags,    /* Context flags indicate Byte mode */
                     0);       /* Timeout */
   if (ccode) goto error;

   /* Get the object name at the current position. */
   ccode = NWDSItrGetCurrent(iter, &iterHandle, pb);
   if (ccode == ERR_EOF_HIT)
      printf("\nNo objects in list were greater or equal to
             typedown string.\n\ Iterator is positioned at EOF\n");
   if (ccode) goto error;


   ccode = NWDSGetObjectCount (context,pb,&objCount);
   if (ccode) goto error;

   ccode = NWDSGetObjectName (context,
                              pb,
                              objName,
                              &attrCount,
                              &objectInfo);
   if (ccode) goto error;

   printf("\nTyped down to object:  %s\n", objName);



/* Clean up, normal or error termination. */
error:
   printf("\n\nccode = %d.",ccode);
   if (iter)
      NWDSItrDestroy(iter);      /* Destroy the Iterator when done. */
   if (pb)
      NWDSFreeBuf(pb);
   NWDSFreeContext(context);
}
```

4.9 Skipping Objects in the List: Example

```
/* itrskip.c
   Example of NDS 8 Iterator NWDSItrSkip function
   .
```

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwcalls.h>

void main(int argc, char *argv[])
{
   NWDSContextHandle context =
       (NWDSContextHandle)ERR_CONTEXT_CREATION;
   nstr8         strAbbreviatedName[MAX_DN_CHARS+1];
   NWDSCCODE     ccode;
   nuint32       flags;
   nuint32       iter=0;
   nint32        iterHandle = -1;
   nuint32       timeout = 0;   /* Timeout=0 means no time limit. */
   nint32        numToSkip = 1;
   nint32        actual;

/* Check arguments */
   if(argc < 2)
   {
      printf("\nUsage:    itrskip  <base object>  <+/-skip count>\n");
      printf("\nExample:  itrskip  myou.myorg  -100\n");
      exit(1);
   }

   if (argc >= 3)
      sscanf(argv[2], "%d", &numToSkip);

   /* Initialize NWCalls and unicode tables */
   ccode = NWCallsInit(NULL,NULL);
      if(ccode) exit(1);

   /* Create NDS Context */
   ccode = NWDSCreateContextHandle(&context);
   if(ccode) exit(1);

   /* Set flags to get object names in typeless format. */
   NWDSGetContext(context, DCK_FLAGS, &flags);
   flags |= DCV_TYPELESS_NAMES;
   NWDSSetContext(context, DCK_FLAGS, &flags);

   printf("itrskip:   Sample program for NWDSItrCount\n");
   printf("Container:    %s\n", argv[1]);
   printf("Number of objects to skip:  %d\n\n", numToSkip);

   /* Convert the directory name (passed in as first arg) to its
      shortest form relative to the name context */
   ccode = NWDSAbbreviateName(context, argv[1], strAbbreviatedName);
   if(ccode) goto error;

   /* ——— Create the Iterator ———- */
```

```
   ccode = NWDSItrCreateList(
                context,              /* context for this search */
                strAbbreviatedName,   /* container to search     */
                NULL,                 /* class name filter       */
                NULL,                 /* subordinate name filter */
                DS_ITR_ANY_SERVER,    /* scalability requirement */
                timeout,              /* 0 = no timeout          */
                &iter);               /* returned Iterator ptr   */
   if (ccode)
      goto error;


   printf("Skipping %d objects...\n", numToSkip);
   ccode = NWDSItrSkip(iter, numToSkip, timeout, &actual);
   if (ccode == 0)
      printf("Actual number of objects skipped = %d\n", actual);
   else if (ccode == ERR_EOF_HIT)
      printf("You tried to skip forward when Iterator was already
              at EOF.\n");
   else if (ccode == ERR_BOF_HIT)
      printf("You tried to skip backward when Iterator was already
              at top.\n");

/* Clean up, normal or error termination. */
error:
   printf("\n\nccode = %d.",ccode);
   if (iter)
      NWDSItrDestroy(iter);      /* Destroy the Iterator when done. */
   NWDSFreeContext(context);
}
```

Revision History

A

The following table lists all changes made to the Novell® eDirectory™ Iterator Services documentation:

February 2008	To accommodate 64-bit platforms, changed the data type of the iteration handle from nuint32 to nuint_ptr. The typedefs from ntypes.h handles this change for backwards compatibility with older 32-bit applications.
March 1, 2006	Added navigational links.
October 5, 2005	Transitioned to revised Novell documentation standards.
February 2004	Renamed the product name from “NDS” to “Novell eDirectory” at relevant instances.
May 1999	Added sample code.
March 1999	Added to the NDK.