Novell Developer Kit

www.novell.com

BINDERY MANAGEMENT

March 1, 2006

Novell®

Legal Notices

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

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

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

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

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

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

Novell Trademarks

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.
Beagle 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.
Hula 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 registered trademark of Novell, Inc., in the United States.
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 Novell, Inc., in the United States and other countries.
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.

Contents

About This Guide

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

Feedback

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

Documentation Updates

For the most recent version of this guide, see NLM and NetWare Libraries for C (including CLIB and XPlat) (http://developer.novell.com/ndk/clib.htm).

Additional Information

For information about other CLib and XPlat interfaces, see the following guides:

- *NDK: NLM Development Concepts, Tools, and Functions*
- *NDK: Program Management*
- *NDK: NLM Threads Management*
- *NDK: Connection, Message, and NCP Extensions*
- *NDK: Multiple and Inter-File Services*
- *NDK: Single and Intra-File Services*
- *NDK: Volume Management*
- *NDK: Client Management*
- *NDK: Network Management*
- *NDK: Server Management*
- *NDK: Internationalization*
- *NDK: Unicode*
- *NDK: Sample Code*
- *NDK: Getting Started with NetWare Cross-Platform Libraries for C*

For CLib source code projects, visit Forge (http://forge.novell.com).

For help with CLib and XPlat problems or questions, visit the NLM and NetWare Libraries for C (including CLIB and XPlat) Developer Support Forums (http://developer.novell.com/ndk/devforums.htm). There are two for NLM development (XPlat and CLib) and one for Windows XPlat development.

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.

Bindery Concepts

1

This documentation describes Bindery, its functions, and features.

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

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

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

1.1 Bindery vs. NDS

The NetWare® 4.x OS replaced the bindery with NDS® (an object database). NDS offers many advantages over the bindery, including a hierarchical structure and global naming.

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

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

1.2 Bindery Files

Each NetWare 3.x server maintains a bindery database. One server cannot coordinate its bindery data with other servers on the network. For NetWare 3.11, the bindery consists of three files:

- net$obj.sys
- net$prop.sys
- net$val.sys

Bindery files are hidden in the SYS:SYSTEM directory.

Bindery allows applications to open and close the bindery so that these files can be archived, but an application must have supervisor equivalence to open and close the bindery. When the files are closed, many server operations become disabled so take precautions. In NetWare 4.x, 5.x, and 6.x, the bindery, which is a bindery emulation service, cannot be opened or closed.

1.2.1 Activity Coordination

Bindery makes no attempt to coordinate activities among multiple stations that concurrently read or write data to a single property. One workstation might read a partially updated property and get inconsistent data if the data of the property extends across multiple segments. Coordination on

Reads and Writes must be handled by application programs. Logical record locks can be used to coordinate activities among applications.

1.3 Bindery Objects

All bindery objects must be assigned the following information:

- Object ID
- Object Type
- Object Name
- Object Flags
- Object Security
- Has-Properties Flag

1.3.1 Object ID

The `objectID` parameter is a 4-byte number uniquely identifying the object within a particular server bindery and is not recognized by other servers. The NetWare OS (not the application) assigns this number.

1.3.2 Object Type

The `objectType` parameter is a 2-byte number classifying the object. Do not swap the following values since they are defined the way they are expected by various functions:

Table 1-1 *Standard Object Types*

C	Pascal	Name
0xFFFF	$FFFF	OT_WILD
0x0000	$0000	OT_UNKNOWN
0x0100	$0100	OT_USER
0x0200	$0200	OT_USER_GROUP
0x0300	$0300	OT_PRINT_QUEUE
0x0400	$0400	OT_FILE_SERVER
0x0500	$0500	OT_JOB_SERVER
0x0600	$0600	OT_GATEWAY
0x0700	$0700	OT_PRINT_SERVER
0x0800	$0800	OT_ARCHIVE_QUEUE
0x0900	$0900	OT_ARCHIVE_SERVER
0x0A00	$0A00	OT_JOB_QUEUE
0x0B00	$0B00	OT_ADMINISTRATION

C	Pascal	Name
0x2100	$2100	OT_NAS_SNA_GATEWAY
0x2600	$2600	OT_REMOTE_BRIDGE_SERVER
0x2700	$2700	OT_TCPIP_GATEWAY

1.3.3 Object Name

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

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

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

1.3.4 Object Flags

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

00h Static
01h Dynamic

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

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

1.3.5 Object Security

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

Table 1-2 *Bindery Object Security Levels*

Bit	Security Level	Description
0x00	BS_ANY_READ	Anyone can read the object, even users who aren't logged in.
0x01	BS_LOGGED_READ	Only clients logged in to the server can read the object.

Bit	Security Level	Description
0x02	BS_OBJECT_READ	Only clients logged in to the server with this object's name, type, and password can read the object.
0x03	BS_SUPER_READ	Only clients with supervisor equivalence can read the object.
0x04	BS_BINDERY_READ	Only the NetWare operating system can read the object.
0x00	BS_ANY_WRITE	Anyone can modify the object, even users who aren't logged in.
0x10	BS_LOGGED_WRITE	Only clients logged in to the server can modify the object.
0x20	BS_OBJECT_WRITE	Only clients logged in to the server with this object's name, type, and password can modify the object.
0x30	BS_SUPER_WRITE	Only clients with supervisor equivalence can modify the object.
0x40	BS_BINDERY_WRITE	Only the NetWare operating system can modify the object.

1.3.6 Has-Properties Flag

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

00h No properties
FFh 1 or more properties

1.4 Bindery Object Properties

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

- Property Name
- Property Flags
- Property Security

1.4.1 Property Name

The `propertyName` parameter is a 15-byte string (including a NULL terminator) containing the name of the property.

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

1.4.2 Property Flags

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

- Bit 0:

0 Static
1 Dynamic

- Bit 1:

0 Item
1 Set

Static and Dynamic Properties

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

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

Item Property

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

Set Property

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

1.4.3 Property Security

The `property security` parameter is a single-byte flag that determines who can access the property. The low-order nibble determines who can scan for and find the property (read security). The high-order nibble determines who can add values to the property (write security). Possible values for this flag correspond to those defined for the object security field. See the table in "Object Security" on page 15.

1.5 Standard Bindery Properties

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

Table 1-3 *Standard Bindery Properties*

Hex	Name	Flags	Object Type: Description
0x32	ACCOUNT_BALANCE	static/item	user
0x32	ACCOUNT_HOLDS	dynamic/item	user
0x31	ACCOUNT_SERVERS	static/set	server

Hex	Name	Flags	Object Type: Description
0x33	ACCT_LOCKOUT	static/item	server
0x31	BLOCKS_READ	static/item	server
0x31	BLOCKS_WRITTEN	static/item	server
0x31	CONNECT_TIME	static/item	server
0x31	DISK_STORAGE	static/item	server
0x31	GROUP_MEMBERS	static/set	user group: List of users that are members of a user group
0x31	GROUPS_I'M_IN	static/set	user: List of user groups of which a user is a member
0x31	IDENTIFICATION	static/item	user: User of users group full name
0x32	LOGIN_CONTROL	static/item	user
0x40	NET_ADDRESS	dynamic/item	server
0x32	NODE_CONTROL	static/item	user
0x33	OLD_PASSWORDS	static/item	user
0x33	OPERATORS	static/set	server: List of objects that are console operators
0x44	PASSWORD	static/item	user: Encrypted password of an object
0x33	Q_DIRECTORY	static/item	queue
0x31	Q_OPERATORS	static/set	queue
0x31	Q_SERVERS	static/set	queue
0x31	Q_USERS	static/set	queue
0x31	REQUESTS_MADE	static/item	server
0x32	SECURITY_EQUALS	static/set	user: List of objects that are equivalent to the associated object
0x31	USER_DEFAULTS	static/item	supervisor

1.6 Bindery Properties Associated with NetWare Security

The following properties are associated with NetWare security:

- LOGIN_CONTROL
- OLD_PASSWORDS
- NODE_CONTROL
- ACCT_LOCKOUT

1.6.1 USER_DEFAULTS and LOGIN_CONTROL Properties

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

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

	Size	Field: Description
0	nuint8[3]	account expiration date: Date the account expires
3	nuint8	account disabled flag: Whether the account is enabled (0x00) or disabled (0xFF). NetWare checks the value every half hour. If set to 0xFF since a user logged into an account, NetWare asks the user to log out. Within five minutes, NetWare will clear the connection.
4	nuint8[3]	password expiration date: Date the password expires
7	nuint8	grace logins remaining: Number of times a user can log in using an expired password. If set to 0xFF, it is not decremented. Otherwise, it is decremented each time the user logs in after the password expired. If set to 0, the user cannot log in.
8	nuint16	password expiration interval: Number of days between password expiration dates. In 4.x, 5.x, and 6.x, it must be nonzero to check the old password list when changing a password.
10	nuint8	grace login reset value: Reset value for the grace logins remaining field after a password change
11	nuint8	minimum password length: Minimum length permitted for a password. If 0, no password is needed. In 4.x, 5.x, and 6.x, it must be nonzero to check the old password list when changing a password.
12	nuint16	maximum concurrent connections: Maximum concurrent connections allowed per user. If 0, the limit is the maximum supported by the server.
14	nuint8[42]	allowed login time bit map: 336 half-hour periods during a week when a user can login to the server. Bit 1 of byte 0 represents 12:00 to 12:29 am Sunday. Bit 2 of byte 0 is 12:30 to 12:59 am Sunday. Setting the bit permits logins during the corresponding period.
56	nuint[6]	last login date and time: Most recent time the user logged in
62	nuint8	restriction flags: Who is allowed to change the password on the account: 0x00 Anyone, 0x01 Supervisor. 0x02 is set if the OLD_PASSWORDS property exists for the account. In 3.x, 4.x, 5.x, and 6.x, it must be 0x02 to keep an old passwords list.
63	nuint8	reserved
64	nuint32	maximum disk usage (in blocks): Number of 4K disk blocks that may be used by the account. If 0x7FFFFFFF, there is no limit.
68	nuint16	bad login count: Number of bad login attempts since the last reset time. (Intruder detection is active only if the ACCT_LOCKOUT property is assigned to the file server object). The field is reset when a successful login occurs or the number of minutes in the `reset minutes` parameter of ACCT_LOCKOUT expires. If the account is locked because of intruder detection, it is set to 0xFFFF.

	Size	Field: Description
70	nuint32	next reset time: Time when the bad login count should be set to 0
74	nuint8[12]	bad login address: Address of the station making the last bad login attempt or provoked an account lockout

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

1.6.2 OLD_PASSWORDS Property

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

1.6.3 NODE_CONTROL Property

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

1.6.4 ACCT_LOCKOUT Property

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

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

1.7 Types of Bindery Functions

The types of Bindery Functions follow:

- Bindery Status Functions
- Bindery Object Functions
- Bindery Object Information Functions
- Bindery Property Functions
- Bindery Password Functions

Note that Bindery parameters accept wildcard characters. The asterisk matches 0 or more characters while the question mark matches exactly one character, as illustrated in the following table:

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

1.7.1 Bindery Status Functions

Bindery Status Functions open and close the bindery files on a specified server and are not supported by NetWare 4.x, 5.x, and 6.x bindery emulation services:

NWCloseBindery (page 38)
NWOpenBindery (page 70)

1.7.2 Bindery Object Functions

These functions perform operations on bindery objects:

NWChangeObjectSecurity (page 33)
NWCreateObject (page 40)
NWDeleteObject (page 46)
NWRenameObject (page 75)
NWScanObject (page 77)

1.7.3 Bindery Object Information Functions

These functions return information related to bindery objects:

NWGetBinderyAccessLevel (page 54)
NWGetObjectDiskSpaceLeft (page 56)
NWGetObjectEffectiveRights (page 60)
NWGetObjectID (page 58)
NWGetObjectName (page 65)
NWScanObjectTrusteePaths (page 80)

1.7.4 Bindery Property Functions

These functions operate on bindery properties:

NWAddObjectToSet (page 28)
NWChangePropertySecurity (page 35)
NWCreateProperty (page 43)
NWDeleteObjectFromSet (page 48)
NWDeleteProperty (page 50)
NWIsObjectInSet (page 67)
NWReadPropertyValue (page 72)

NWScanProperty (page 85)
NWWritePropertyValue (page 90)

1.7.5 Bindery Password Functions

These functions perform operations on object passwords:

NWChangeObjectPassword (page 31)
NWDisallowObjectPassword (page 52)
NWVerifyObjectPassword (page 88)

Bindery Tasks

2

This documentation describes common tasks associated with Bindery.

2.1 Creating a Bindery Object

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

To create an object:

1. Call the NWCreateObject function.
2. Call the NWCreateProperty function.
3. Call the NWWritePropertyValue function.

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

2.2 Scanning for Bindery Objects

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

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

"*" Matches any object name.

"S*" Matches any object name beginning with S.

"??" Matches any two-character object name.

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

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

See binscan.c (../../../samplecode/clib_sample/bindery/binscan/binscan.c.html) for sample code.

2.3 Scanning Bindery Properties

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

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

1. On the initial call, set the object name and type appropriately, and set the search property name to the asterisk wild card.
2. On the initial call, also set the `iterHandle` parameter to –1.
3. When the function returns, check the `moreFlag` parameter.
   - If the `moreFlag` parameter is set to 0xFF, there are more properties. The `iterHandle` parameter will be set in preparation for the next call to the NWScanProperty function.
   - If there are no more properties, the `moreFlag` parameter is set to 0.

See binscan.c (../../../samplecode/clib_sample/bindery/binscan/binscan.c.html) for sample code.

2.4 Reading the Value of a Bindery Property

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

See binscan.c (../../../samplecode/clib_sample/bindery/binscan/binscan.c.html) for sample code.

2.5 Checking for a Member of a Set Property

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

2.6 Setting Bindery Emulation

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

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

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

```
set Bindery Context = o = MyOrg
```

(where "MyOrg" is a organization having a valid replica on the server being set).

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

NOTE: It is possible to enter the command with an invalid organization. The command returns a message that bindery context is set to the invalid organization, but only the bindery string is set-bindery emulation remains unset. Any calls to bindery functions then fail and return errors.

To verify that bindery emulation is actually set, at the server command prompt, type config. The displayed configuration includes a report of all valid bindery emulation settings.

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

Bindery Functions

3

This documentation alphabetically lists the Bindery functions and describes their purpose, syntax, parameters, and return values.

NWAddObjectToSet

Adds a member to a bindery property of type SET

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API  NWAddObjectToSet  (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   const nstr8 N_FAR  *propertyName,
   const nstr8 N_FAR  *memberName,
   nuint16             memberType);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

objName

(IN) Points to the new SET object name.

objType

(IN) Specifies the SET object type.

propertyName

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

memberName

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

memberType

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89E8	WRITE_PROPERTY_TO_GROUP
0x89E9	MEMBER_ALREADY_EXISTS
0x89EA	NO_SUCH_MEMBERS
0x89EB	NOT_GROUP_PROPERTY
0x89EC	NO_SUCH_SEGMENT
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F8	NO_PROPERTY_WRITE_PRIVILEGE
0x89FB	NO_SUCH_PROPERTY
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

NCP Calls

0x2222 23 65 Add Bindery Object To Set

See Also

NWDeleteObjectFromSet (page 48), NWIsObjectInSet (page 67)

NWChangeObjectPassword

Changes the specified password of an object to a new password

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API  NWChangeObjectPassword  (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   const nstr8 N_FAR  *oldPassword,
   const nstr8 N_FAR  *newPassword);
```

Pascal

```
uses calwin32

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

Parameters

conn

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

objName

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

objType

(IN) Specifies the type of the object.

oldPassword

(IN) Points to the old password.

newPassword

(IN) Points to the new password.

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x89D7	PASSWORD_NOT_UNIQUE
0x89D8	PASSWORD_TOO_SHORT
0x89DC	ACCOUNT_DISABLED
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE
0x89F0	WILD_CARD_NOT_ALLOWED
0x89FB	INVALID_PARAMETER
0x89FF	NO_SUCH_OBJECT_OR_BAD_PASSWORD

Remarks

NWChangeObjectPassword does not require the old password to be known.

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

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

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

See "Object Type" on page 14.

NCP Calls

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

See Also

NWCreateObject (page 40), NWCreateProperty (page 43)

NWChangeObjectSecurity

Changes the security access mask of a bindery object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWChangeObjectSecurity (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   nuint8              newObjSecurity);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

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

objType

(IN) Specifies the bindery object type.

newObjSecurity

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F1	INVALID_BINDERY_SECURITY
0x89F5	NO_OBJECT_CREATE_PRIVILEGE
0x89FC	NO_SUCH_OBJECT
0x98FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

See Section 4.3, “Security Rights Mask Values,” on page 93.

See “Object Type” on page 14.

See Section 4.1, “Extended Object Type Values,” on page 93.

NCP Calls

0x2222 23 56 Change Bindery Object Security

See Also

NWGetObjectID (page 58)

NWChangePropertySecurity

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWChangePropertySecurity (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   const nstr8 N_FAR  *propertyName,
   nuint8              newPropertySecurity);
```

Pascal

```
uses calwin32

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

Parameters

conn

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

objName

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

objType

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

propertyName

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

newPropertySecurity

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F1	INVALID_BINDERY_SECURITY
0x89F2	NO_OBJECT_READ_PRIVILEGE
0x89F5	NO_OBJECT_CREATE_PRIVILEGE
0x89F6	NO_PROPERTY_DELETE_PRIVILEGE
0x89FC	NO_SUCH_PROPERTY
0x89FC	NO_SUCH_OBJECT
0x98FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

See Section 4.3, “Security Rights Mask Values,” on page 93.

See “Object Type” on page 14.

See Section 4.1, “Extended Object Type Values,” on page 93.

NCP Calls

0x2222 23 59 Change Property Security

See Also

NWChangeObjectSecurity (page 33)

NWCloseBindery

Closes the bindery

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API  NWCloseBindery  (
   NWCONN_HANDLE   conn);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

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

NCP Calls

0x2222 23 68 Close Bindery

See Also

NWOpenBindery (page 70)

NWCreateObject

Creates a bindery object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

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

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

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

objType

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

objFlags

(IN) Specifies whether the new object is dynamic:

BF_DYNAMIC
BF_STATIC

objSecurity

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89E7	E_NO_MORE_USERS
0x89EE	OBJECT_ALREADY_EXISTS
0x89EF	INVALID_NAME
0x89F1	INVALID_BINDERY_SECURITY
0x89F5	NO_OBJECT_CREATE_PRIVILEGE
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

NWCreateObject requires SUPERVISOR or equivalent rights.

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

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

See Section 4.3, “Security Rights Mask Values,” on page 93.

See “Object Type” on page 14.

NCP Calls

0x2222 23 50 Create Bindery Object

See Also

NWChangeObjectPassword (page 31), NWCreateProperty (page 43)

NWCreateProperty

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API  NWCreateProperty  (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   const nstr8 N_FAR  *propertyName,
   nuint8              propertyFlags,
   nuint8              propertySecurity);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

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

objType

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

propertyName

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

propertyFlags

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

BF_DYNAMIC
BF_STATIC

propertySecurity

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89ED	PROPERTY_ALREADY_EXISTS
0x89EF	INVALID_NAME
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F1	INVALID_BINDERY_SECURITY
0x89F2	NO_OBJECT_READ_PRIVILEGE
0x89F6	NO_PROPERTY_DELETE_PRIVILEGE
0x89F7	NO_PROPERTY_CREATE_PRIVILEGE
0x89FB	N0_SUCH_PROPERTY
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

NWCreateProperty requires Write access to the bindery object.

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

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

See Section 4.3, “Security Rights Mask Values,” on page 93.

See “Object Type” on page 14.

NCP Calls

0x2222 23 57 Create Property

See Also

NWChangeObjectPassword (page 31), NWCreateObject (page 40)

NWDeleteObject

Deletes a bindery object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWDeleteObject (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the object name being deleted.

objType

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F4	NO_OBJECT_DELETE_PRIVILEGE
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

NWDeleteObject requires SUPERVISOR or equivalent rights.

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

See "Object Type" on page 14.

NCP Calls

0x2222 23 51 Delete Bindery Object

See Also

NWDeleteObjectFromSet (page 48)

NWDeleteObjectFromSet

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWDeleteObjectFromSet (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objectName,
   nuint16             objectTpe,
   pnstr8              propertyName,
   pnstr8              memberName,
   nuint16             memberType);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objectName

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

objectType

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

propertyName

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

memberName

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

memberType

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89EA	NO_SUCH_MEMBER
0x89EB	NOT_GROUP_PROPERTY
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F4	NO_OBJECT_DELETE_PRIVILEGE
0x89F8	NO_PROPERTY_WRITE_PRIVILEGE
0x89FB	N0_SUCH_PROPERTY
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

See “Object Type” on page 14.

NCP Calls

0x2222 23 66 Delete Bindery Object From Set

See Also

NWDeleteObject (page 46), NWDeleteProperty (page 50)

NWDeleteProperty

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWDeleteProperty (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objectName,
   nuint16             objectType,
   const nstr8 N_FAR  *propertyName);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objectName

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

objectType

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

propertyName

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F1	INVALID_BINDERY_SECURITY
0x89F6	NO_PROPERTY_DELETE_PRIVILEGE
0x89FB	NO_SUCH_PROPERTY
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

See “Object Type” on page 14.

NCP Calls

0x2222 23 58 Delete Property

See Also

NWDeleteObjectFromSet (page 48), NWDeleteObject (page 46)

NWDisallowObjectPassword

Prevents use of the specified password by the specified object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE NWAPI  NWDisallowObjectPassword  (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   const nstr8 N_FAR  *disallowedPassword);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

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

objType

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

disallowedPassword

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x00FF	BINDERY_FAILURE
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x89F0	WILD_CARD_NOT_ALLOWED
0x89FB	INVALID_PARAMETERS
0x89FC	NO_SUCH_OBJECT
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

See "Object Type" on page 14.

NCP Calls

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

See Also

NWChangeObjectPassword (page 31), NWLoginToFileServer (Server Management)

NWGetBinderyAccessLevel

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

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

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

accessLevel

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

objID

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY

Remarks

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

See Section 4.3, “Security Rights Mask Values,” on page 93.

NCP Calls

0x2222 23 70 Get Bindery Access Level

NWGetObjectDiskSpaceLeft

Returns the remaining disk space for a specified object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

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

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

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

systemElapsedTime

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

unusedDiskBlocks

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

restrictionEnforced

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

0x0000 Enforced
0x00FF Not enforced

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x89C6	NO_CONSOLE_PRIVILEGES
0x89FC	NO_SUCH_OBJECT

Remarks

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

NCP Calls

0x2222 23 230 Get Object's Remaining Disk Space

NWGetObjectID

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API  NWGetObjectID  (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   pnuint32            objID);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

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

objType

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

objID

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F1	INVALID_BINDERY_SECURITY
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

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

See "Object Type" on page 14.

NCP Calls

0x2222 23 53 Get Bindery Object ID

See Also

NWChangeObjectSecurity (page 33), NWCreateObject (page 40)

NWGetObjectEffectiveRights

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

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 NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWGetObjectEffectiveRights (
   NWCONN_HANDLE       conn,
   nuint32             objID,
   NWDIR_HANDLE        dirHandle,
   const nstr8 N_FAR  *path,
   pnuint16            rightsMask);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

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

dirHandle

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

path

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

rightsMask

(OUT) Points to the rights mask.

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A8	ERR_ACCESS_DENIED
0x89FB	INVALID_PARAMETERS
0x89FC	NO_SUCH_OBJECT

Remarks

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

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

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

See Section 4.2, “Maximum Rights Mask Values,” on page 93.

NCP Calls

0x2222 22 50 Get Object Effective Rights For Directory Entry

See Also

NWGetEffectiveRights (Multiple and Inter-File Management)

NWGetObjectEffectiveRightsExt

Returns the effective rights of an object to the specified directory or file, using UTF-8 strings.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 6.5 SP2 or later

Platform: NLM, Windows 2000, Windows XP

Client: 4.90 SP2 or later

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

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

NWCCODE N_API NWGetObjectEffectiveRightsExt (
   NWCONN_HANDLE       conn,
   nuint32             objID,
   NWDIR_HANDLE        dirHandle,
   const nstr8 N_FAR  *path,
   nuint8              buNameSpace,
   pnuint16            rightsMask);
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

(IN) Specifies the ID of the object of the user.

dirHandle

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

path

(IN) Points to the absolute path (or a path relative to the `dirhandle` parameter) of the directory or file whose effective rights mask is being reported. The characters in the string must be UTF-8.

buNameSpace

(IN) Specifies the name space of the specified path. See Name Space Flag Values in *Multiple and Inter-File Services*.

rightsMask

(OUT) Points to the rights mask. See Inherited Rights Mask Values. If the rightsMask parameter is zero, the client has no rights in the target directory.

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88F0	UTF8_CONVERSION_FAILED
0x890A	NLM_INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A8	ERR_ACCESS_DENIED
0x89FB	INVALID_PARAMETERS
0x89FC	NO_SUCH_OBJECT

Remarks

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

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

NOTE: For information about the requirments for using the NWGetObjectEffectiveRightsExt function, see UTF-8 Path and Filenames in *Multiple and Inter-File Services*.

NCP Calls

0x2222 22 50 Get Object Effective Rights For Directory Entry
0x2222 89 50 Enhanced Get Object Effective Rights

See Also

NWGetEffectiveRightsExt (Multiple and Inter-File Management)

NWGetObjectName

Returns the name and object type of a bindery object on the network server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

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

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

(IN) Specifies the object ID.

objName

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

objType

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89F1	INVALID_BINDERY_SECURITY
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

All parameter positions must be filled.

See "Object Type" on page 14.

NCP Calls

0x2222 23 54 Get Bindery Object Name

See Also

NWChangeObjectSecurity (page 33), NWCreateObject (page 40), NWGetObjectID (page 58)

NWIsObjectInSet

Searches a property of type SET for a specified object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWIsObjectInSet (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   const nstr8 N_FAR  * propertyName,
   const nstr8 N_FAR  *memberName,
   nuint16             memberType);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

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

objType

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

propertyName

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

memberName

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

memberType

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89EA	NO_SUCH_MEMBER
0x89EB	NOT_GROUP_PROPERTY
0x89EC	NO_SUCH_SEGMENT
0x89F0	WILD_CARD_NOT_ALLOWED
0x89FB	N0_SUCH_PROPERTY

Remarks

NWIsObjectInSet requires Read access to the SET property.

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

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

Object	Property	Property Value
JOAN		
SECRETARIES	GROUP_MEMBERS	The object ID of JOAN
EMPLOYEES	GROUP_MEMBERS	The object ID of SECRETARIES

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

See "Object Type" on page 14.

NCP Calls

0x2222 23 67 Is Bindery Object In Set

See Also

NWAddObjectToSet (page 28), NWDeleteObjectFromSet (page 48)

NWOpenBindery

Reopens a NetWare server bindery closed by calling the NWCloseBindery function

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API  NWOpenBindery  (
   NWCONN_HANDLE   conn);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

NCP Calls

0x2222 23 69 Open Bindery

NWReadPropertyValue

Reads the property value of a bindery object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API  NWReadPropertyValue  (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   const nstr8 N_FAR  *propertyName,
   nuint8              segmentNum,
   pnuint8             segmentData,
   pnuint8             moreSegments,
   pnuint8             flags);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

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

objType

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

propertyName

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

segmentNum

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

segmentData

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

moreSegments

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

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

flags

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

BF_ITEM
BF_SET

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE
0x8993	NO_READ_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x89EC	NO_SUCH_SEGMENT
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F1	INVALID_BINDERY_SECURITY
0x89F9	NO_PROPERTY_READ_PRIVILEGES
0x89FB	N0_SUCH_PROPERTY
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED

0x89FF	HARDWARE_FAILURE

Remarks

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

On the first call to NWReadPropertyValue set the `segmentNum` parameter to 1. For each subsequet call, increment `segmentNum` until the `moreSegments` parameter is set to 0 or until NWReadPropertyValue returns NO_SUCH_SEGMENT.

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

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

See "Object Type" on page 14.

See "Activity Coordination" on page 13.

See binscan.c (../../../samplecode/clib_sample/bindery_binscan/binscan/binscan.c.html) for sample code.

NCP Calls

0x2222 23 61 Read Property Value

See Also

NWWritePropertyValue (page 90)

NWRenameObject

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWRenameObject (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *oldObjName,
   const nstr8 N_FAR  *newObjName,
   nuint16             objType);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

oldObjName

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

newObjName

(IN) Points to the new object name.

objType

(IN) Specifies the type of the object.

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89EE	OBJECT_ALREADY_EXISTS
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F3	NO_OBJECT_RENAME_PRIVILEGE
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

See "Object Type" on page 14.

NCP Calls

0x2222 23 52 Rename Object

See Also

NWScanObject (page 77)

NWScanObject

Searches for a bindery object name

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWScanObject (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *searchName,
   nuint16             searchType,
   pnuint32            objID,
   pnstr8              objName,
   pnuint16            objType,
   pnuint8             hasPropertiesFlag,
   pnuint8             objFlags,
   pnuint8             objSecurity);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

searchName

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

searchType

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

objID

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

objName

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

objType

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

hasPropertiesFlag

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

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

objFlags

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

BF_STATIC
BF_DYNAMIC

objSecurity

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89EF	INVALID_NAME
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

All parameter positions must be filled.

See Section 4.3, “Security Rights Mask Values,” on page 93.

See “Object Type” on page 14.

See binscan.c (../../../samplecode/clib_sample/bindery_binscan/binscan/binscan.c.html) for sample code.

NCP Calls

0x2222 23 55 Scan Bindery Object

NWScanObjectTrusteePaths

Returns the directory paths to which an object has trustee rights

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

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

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

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

volNum

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

iterHandle

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

accessRights

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

dirPath

(OUT) Points to the directory path name in the DOS name space. This buffer should be at least 270 bytes.

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899C	NO_MORE_TRUSTEES
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F2	NO_OBJECT_READ_PRIVILEGE
0x89FC	NO_SUCH_OBJECT

Remarks

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

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

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

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

NWScanObjectTrusteePaths was originally written for the 2.x platform and does not handle 3.x, 4.x, 5.x, and 6.x rights perfectly. For example, NWScanObjectTrusteePaths does not return the 2.x "Supervisory" right. To retrieve the correct trustee rights on the 3.x, 4.x, 5.x, and 6.x platforms, call NWScanObjectTrusteePaths to obtain a path. Then call the NWIntScanForTrustees function to return the rights of the object to the path.

See Section 4.2, “Maximum Rights Mask Values,” on page 93.

NCP Calls

0x2222 23 71 Scan Bindery Object Trustee Paths

See Also

NWIntScanForTrustees (Multiple and Inter-File Management)

NWScanObjectTrusteePathsExt

Returns the directory paths to which an object has trustee rights, using UTF-8 strings.

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 6.5 SP2 or later

Platform: NLM, Windows 2000, Windows XP

Client: 4.90 SP2 or later

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

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

NWCCODE N_API  NWScanObjectTrusteePathsExt  (
   NWCONN_HANDLE   conn,
   nuint32         objID,
   nuint16         volNum,
   pnuint16        iterHandle,
   pnuint8         accessRights,
   pnstr8          dirPath1506);
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objID

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

volNum

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

iterHandle

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

accessRights

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

dirPath1506

(OUT) Points to the directory path name in the DOS name space. This buffer should be at least 1506 bytes.

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88F0	UTF8_CONVERSION_FAILED
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899C	NO_MORE_TRUSTEES
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F2	NO_OBJECT_READ_PRIVILEGE
0x89FC	NO_SUCH_OBJECT

Remarks

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

NOTE: For information about the requirments for using the NWScanObjectTrusteePathsExt function, see UTF-8 Path and Filenames in *Multiple and Inter-File Services*.

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

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

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

NWScanObjectTrusteePathsExt does not handle 6.x rights perfectly. To retrieve the correct trustee rights on the 6.x platforms, call NWScanObjectTrusteePathsExt to obtain a path. Then call the NWIntScanForTrusteesExt function in *Multiple and Inter-File Services* to return the rights of the object to the path.

NCP Calls

0x2222 23 71 Scan Bindery Object Trustee Paths
0x2222 89 71 Enhanced Scan Volume Trustee Object Paths

See Also

NWIntScanForTrusteesExt (Multiple and Inter-File Management)

NWScanProperty

Scans the given bindery object for properties matching the property name

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API NWScanProperty (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   pnstr8              searchPropertyName,
   pnuint32            iterHandle,
   pnstr8              propertyName,
   pnuint8             propertyFlags,
   pnuint8             propertySecurity,
   pnuint8             valueAvailable,
   pnuint8             moreFlag);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

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

objType

(IN) Specifies the object type of the set.

searchPropertyName

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

iterHandle

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

propertyName

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

propertyFlags

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

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

propertySecurity

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

valueAvailable

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

0x00 Property has no value
0xFF Property has value

moreFlag

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

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL

0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89F0	WILD_CARD_NOT_ALLOWED
0x89FB	NO_SUCH_PROPERTY
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

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

See Section 4.3, “Security Rights Mask Values,” on page 93.

See “Object Type” on page 14.

See binscan.c (../../../samplecode/clib_sample/bindery_binscan/binscan/binscan.c.html) for sample code.

NCP Calls

0x2222 23 60 Scan Property

See Also

NWReadPropertyValue (page 72), NWWritePropertyValue (page 90)

NWVerifyObjectPassword

Verifies the password of a bindery object on the specified NetWare server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API  NWVerifyObjectPassword  (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *objName,
   nuint16             objType,
   const nstr8 N_FAR  *password);
```

Pascal

```
uses calwin32

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

Parameters

conn

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

objName

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

objType

(IN) Specifies the type of object (see “Object Type” on page 14 for type values).

password

(IN) Points to the password to be verified.

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x89F0	WILD_CARD_NOT_ALLOWED
0x89FB	INVALID_PARAMETERS
0x89FC	NO_SUCH_OBJECT
0x89FF	NO_SUCH_OBJECT_OR_BAD_PASSWORD

Remarks

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

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

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

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

NCP Calls

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

See Also

NWLoginToFileServer (Server Management)

NWWritePropertyValue

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

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

Library: Cross-Platform NetWare Calls (CAL*.*)

Service: Bindery

Syntax

C

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

NWCCODE N_API  NWWritePropertyValue  (
   NWCONN_HANDLE   conn,
   const nstr8 N_FAR   *objName,
   nuint16              objType,
   const nstr8 N_FAR   *propertyName,
   nuint8               segmentNum,
   const nuint8 N_FAR  *segmentData,
   nuint8               moreSegments);
```

Pascal

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

(IN) Points to the name of the object.

objType

(IN) Specifies the type of the object.

propertyName

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

segmentNum

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

segmentData

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

moreSegments

(IN) Specifies whether more segments are being written:

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

Return Values

These are common return values. See Return Values (*Return Values for C*).

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89E8	WRITE_PROPERTY_TO_GROUP
0x89EC	NO_SUCH_SEGMENT
0x89F0	WILD_CARD_NOT_ALLOWED
0x89F8	NO_PROPERTY_WRITE_PRIVILEGE
0x89FB	N0_SUCH_PROPERTY
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

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

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

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

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

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

See "Object Type" on page 14.

See "Activity Coordination" on page 13.

NCP Calls

0x2222 23 62 Write Property Value

See Also

NWReadPropertyValue (page 72)

Bindery Values

4

This documentation lists defined values associated with Bindery.

4.1 Extended Object Type Values

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

C	Pascal	Values
0x2D00	$2D00	OT_TIME_SYNCHRONIZATION_SERVER
0x2E00	$2E00	OT_ARCHIVE_SERVER_DYNAMIC_SAP
0x4700	$4700	OT_ADVERTISING_PRINT_SERVER
0x5000	$5000	OT_BTRIEVE_VAP
0x5300	$5300	OT_PRINT_QUEUE_USER

4.2 Maximum Rights Mask Values

TA_OPEN is obsolete in 3.x and above.

Hex	Bit
0x00	TA_NONE
0x01	TA_READ
0x02	TA_WRITE
0x04	TA_OPEN
0x08	TA_CREATE
0x10	TA_DELETE
0x20	TA_OWNERSHIP
0x40	TA_SEARCH
0x80	TA_MODIFY
0xFB	TA_ALL

4.3 Security Rights Mask Values

Read Value	Write Value	Access: Description
BS_ANY_READ	BS_ANY_WRITE	Anyone: Access allowed to all clients, even if the client has not logged in to the server

Read Value	Write Value	Access: Description
BS_LOGGED_READ	BS_LOGGED_WRITE	Logged: Access allowed to all clients logged in to the server
BS_OBJECT_READ	BS_OBJECT_WRITE	Object: Access allowed only to clients who have logged in to the server with the name, type, and password of the object
BS_SUPER_READ	BS_SUPER_WRITE	SUPERVISOR: Access allowed only to clients who have logged in to the server as SUPERVISOR, or as a bindery object that is security-equivalent to SUPERVISOR
BS_BINDERY_READ	BS_BINDERY_WRITE	NetWare: Access allowed only to NetWare

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

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

Server-Based Bindery Concepts

5

This documentation describes Server-Based Bindery, its functions, and features.

Each NetWare® 3.x server includes a database, called the bindery, which is currently implemented as three hidden files (NET$OBJ.SYS, NET$PROP.SYS, and NET$VAL.SYS) located in the SYS:SYSTEM directory. The NetWare OS maintains a list in the bindery files of all objects (entities) allowed to access the server. NetWare also records information about each object in the bindery.

NOTE: For NetWare 4.x, the bindery has been replaced by the Directory. However, the Directory can simulate the bindery if bindery context is set.

5.1 Objects

A bindery object can be a user, user group, server, print server, or any other named entity that might gain access to a server. Each bindery object consists of the following components: object ID, object type, object name, object flag, object security, and properties flag.

The NetWare operating system (not the programmer) assigns a 4-byte object ID to an object. This number uniquely identifies the object within a particular servers bindery. An object type classifies an object as a user, user group, server, and so on.

The object name is a 48-byte, NULL-terminated string. The name itself can be 1 to 47 characters long. Only printable characters can be used. An object name cannot include spaces or the following characters:

Table 5-1 *Invalid Characters for Object Names*

/	Slash
\	Backslash
:	Colon
;	Semicolon
,	Comma
*	Asterisk
?	Question mark

The object flag specifies whether the object is Static (0x00) or Dynamic (0x01). A Static object exists in a bindery until an application intentionally deletes it with DeleteBinderyObject. A Dynamic object disappears from a servers bindery when the server is rebooted or when it is specifically deleted. (In the case of an object that is a service-advertising server, the object disappears from a bindery when the server ceases to advertise.)

The object security determines who can access the object. The low-order nibble determines who can read (scan for and find) the object. The high-order nibble determines who can write to (add properties to or delete properties from) the object.

NOTE: Security from NLM applications is derived from the current connection number.

The following table lists defined (well-known) object types:

Table 5-2 *Well-Known Object Types*

0xFFFF (-1)	OT_WILD
0x0000	OT_UNKNOWN
0x0001	OT_USER
0x0002	OT_USER_GROUP
0x0003	OT_PRINT_QUEUE
0x0004	OT_FILE_SERVER
0x0005	OT_JOB_SERVER
0x0006	OT_GATEWAY
0x0007	OT_PRINT_SERVER
0x0008	OT_ARCHIVE_QUEUE
0x0009	OT_ARCHIVE_SERVER
0x000A	OT_JOB_QUEUE
0x000B	OT_ADMINISTRATION
0x0024	OT_REMOTE_BRIDGE_SERVER
0x0047	OT_ADVERTISING_PRINT_SERVER
0x004C	OT_NETWARE_SQL
0x8000	Reserved up to

The following table lists values defined for each nibble:

Hex	Binary	Access Level	Description
0	0 0 0 0	Anyone	Access allowed to all clients, even if the client has not logged in to the server
1	0 0 0 1	Logged	Access allowed only to clients who have logged in to the server
2	0 0 1 0	Object	Access allowed only to clients who have logged in to the server with the objects name, type, and password
3	0 0 1 1	Supervisor	Access allowed only to clients who have logged in to the server as the Supervisor or as a bindery object that is security equivalent to the Supervisor

Hex	Binary	Access Level	Description
4	0 1 0 0	NetWare	Access allowed only to the NetWare operating system

For example, 0x31 indicates that any user logged in to the server can find the object, but only the Supervisor can add a property to the object.

The properties flag indicates whether one or more properties are associated with the object. A value of 0x00 indicates that no properties are associated with the object. A value of 0xFF indicates that one or more properties are associated with the object.

All of these six components (object ID, object type, object name, object flag, object security, and properties flag) are essential elements of a bindery object.

5.2 Properties and Values

Each bindery object can have one or more properties associated with it. For example, the object DAN, a user, (object type 0x01) might have associated with it the properties GROUPS_IM_IN, ACCOUNT_BALANCE, and PASSWORD. Note that GROUPS_IM_IN is not the name of a user group to which the object belongs. It is only the name of one category of information associated with that object. In the same way, ACCOUNT_BALANCE is not an actual numerical balance, and PASSWORD is not an actual password. Properties only identify categories of information associated with the object.

Each property has a value associated with it. For example, a value associated with the property GROUPS_IM_IN must be the object ID of a user group to which DAN belongs. The value of the property ACCOUNT_BALANCE must be DAN's current balance. The value of the property PASSWORD must be DAN's login password (for example, COMPILE). Although a property has one value, the value can contain multiple segments, each being 128 bytes long.

Properties fall into one of two categories: Item properties or Set properties. An Item property has associated with it one or more 128-byte segments. For example, the property ACCOUNT_BALANCE is an Item property with an associated value that contains a monetary balance in the first few bytes of a 128-byte string and zeros in the remaining bytes.

A Set property has associated with it a list of 1 to 32 object IDs contained in a 128-byte segment. Each object ID is a long integer (4 bytes). The property GROUPS_IM_IN is a Set property. The 128-byte segment associated with GROUPS_IM_IN contains the object IDs of 1 to 32 user groups to which (in our example) DAN belongs. The values of Set properties are always object IDs grouped into one or more 128-byte segments.

Each property consists of the following components: property name, property flags, property security, and values flag.

The property name can be 1 to 15 characters long. Only printable characters can be used. A property name cannot include spaces or the following characters:

Table 5-3 *Invalid Characters for Property Names*

/	Slash
\	Backslash

:	Colon
;	Semicolon
,	Comma
*	Asterisk
?	Question mark

A list of defined (well-known) properties appears at the end of this section.

The property flags field is a 1-byte field with two bits defined. Bit 0 is the Static/Dynamic flag defined as shown in the following figure:

Figure 5-1 *Static and DynamicFlag*

7	6	5	4	3	2	1	0	
							0	The property is Static
							1	The property is Dynamic

A Static property exists until it is deleted with the DeleteProperty function, or until the object is deleted. A Dynamic property is deleted from the servers bindery when the server is rebooted.

Bit 1 is the Item/Set flag defined as shown in the following figure:

Figure 5-2 *Item and Set Flag*

7	6	5	4	3	2	1	0	
						0		The property is an item
						1		The property is a set

The values of Item properties are defined and interpreted by applications or by APIs. The bindery process interprets the value of a Set property as a series of object ID numbers, each 4 bytes long.

For example, the bit combination shown in the following figure indicates a Static property of type Set:

Figure 5-3 *A Static Set*

7	6	5	4	3	2	1	0
						1	0

The property security determines who can access the property. The low-order nibble determines who can scan for and find the property. The high-order nibble determines who can add value(s) to the property.

For example, 0x31 indicates that any user logged in to the server can find the property, but only the Supervisor can add value(s) to the property.

NOTE: Security from NLM applications is derived from the current connection number.

The values flag indicates whether the property has a value associated with it. The following table lists defined (well-known) properties:

Table 5-4 *Well-Known Properties*

Property Name	Object Type	Property Flags:		Security		API
		Static/ Dynamic	Item/ Set	Write	Read	
ACCOUNT_BALANCE	User	Static	Item	3	2	Accounting
ACCOUNT_HOLDS	User	Dynamic	Item	3	2	Accounting
ACCOUNT_SERVERS	File Server	Static	Set	3	1	Accounting
ACCT_LOCKOUT	File Server	Static	Item	3	3	Bindery
BLOCKS_READ	File Server	Static	Item	3	1	Accounting
BLOCKS_WRITTEN	File Server	Static	Item	3	1	Accounting
CONNECT_TIME	File Server	Static	Item	3	1	Accounting
DISK_STORAGE	File Server	Static	Item	3	1	Accounting
GROUP_MEMBERS	User Group	Static	Set	3	1	Bindery
GROUPS_IM_IN	User	Static	Set	3	1	Bindery
IDENTIFICATION	User	Static	Item	3	1	Bindery
LOGIN_CONTROL	User	Static	Item	3	2	Bindery
NET_ADDRESS	File Server	Dynamic	Item	4	0	Service Adv.
NODE_CONTROL	User	Static	Item	3	2	Bindery
OLD_PASSWORDS	User	Static	Item	3	3	Bindery
OPERATORS	File Server	Static	Set	3	3	Bindery
PASSWORD	User	Static	Item	4	4	Bindery
Q_DIRECTORY	Print Queue	Static	Item	3	3	Queue Man.
Q_OPERATORS	Print Queue	Static	Set	3	1	Queue Man.
Q_SERVERS	Print Queue	Static	Set	3	1	Queue Man.
Q_USERS	Print Queue	Static	Set	3	1	Queue Man.
REQUESTS_MADE	File Server	Static	Item	3	1	Accounting
SECURITY_EQUALS	User	Static	Set	3	2	Bindery
USER_DEFAULTS	Supervisor	Static	Item	3	1	Bindery

5.3 Workgroup Managers

NetWare versions 3.x and higher support the concept of a workgroup manager (property type MANAGERS). Large organizations may need workgroup managers where several groups that share a single server want autonomous control over their own users and data.

The workgroup manager has duties similar to those of the network Supervisor (with some limitations). A workgroup manager can create and delete users and groups, In addition, a workgroup manager can administer data resources, but only for part of the users in the bindery, and only for part of the logical file system.

The MANAGERS Set property can be added to the SUPERVISOR object. Any object that has security rights equivalent to an object listed in the MANAGERS property is considered to be a workgroup manager. The MANAGERS property is read-Supervisor and write-Supervisor. If the MANAGERS property does not exist, no objects are considered to be workgroup managers.

Workgroup managers, like the network Supervisor, are allowed to create objects in the bindery. All newly-created objects have a set property called OBJ_SUPERVISORS. This property is read-object and write-Supervisor, and is initialized to contain a single element: the object ID of the workgroup manager who created the object. If an object does not have the OBJ_SUPERVISORS property, the object is assumed to have been created by the Supervisor.

The meaning of Supervisor access changes in relation to bindery object access. A caller is considered to be the Supervisor of a target object if that caller is security-equivalent to any other object that appears in the target objects OBJ_SUPERVISORS property. An objects Supervisor can do anything to the object that the network Supervisor can do, including reading and modifying Supervisor-level properties, deleting the object, and so on.

Workgroup managers have rights and abilities beyond those of normal users: they can create new objects in the bindery, and they can control login restriction properties of the users that they create (such as login times, password length, and account balances). However, workgroup managers cannot control these parameters for their own account unless they are included in the OBJ_SUPERVISORS property of their own user object.

Workgroup managers have no extra rights in regard to directory access rights; they must be granted rights in the standard manner by an entity (usually the Supervisor) who has sufficient rights to grant access rights to the appropriate directories. And, as is the case with any user, workgroup managers must have parental rights to a directory before they can grant access rights to that directory to other users (including any users that the workgroup manager created).

A user is considered to be a workgroup manager if that user is security-equivalent to any user listed in the MANAGERS property of the Supervisor. A user has Supervisor rights over a specific object if the user is security-equivalent to anything listed in the objects OBJ_SUPERVISORS property. Therefore, a user can implicitly become a specific objects Supervisor without actually becoming a workgroup manager.

The workgroup manager concept is two-tiered: The network Supervisor creates workgroup managers, and workgroup managers create and manage users. Workgroup managers cannot create other workgroup managers; the relationship is not hierarchical. The bindery treats workgroup managers as peers. More complex relationships can be created by controlling the places in the directory tree where workgroup managers are given rights, effectively creating hierarchical relationships in the data that they control.

Workgroup managers supplement, but do not replace, the network Supervisor. The network Supervisor maintains absolute control over the network. The network Supervisor and all objects that are security-equivalent to the Supervisor can still make all the calls that they could before NetWare provided workgroup manager support.

A workgroup manager cannot add an object to a group that is write-Supervisor unless that manager is the Supervisor of both the group and the object. This restriction ensures that a workgroup manager cannot create a new user and then make that user security-equivalent to the system Supervisor.

Workgroup managers can perform the following tasks:

- Clear another station if the object logged into the target station is one of his supervised objects.
- Create new queues and destroy queues that they have created.
- Add users (objects) to a queue only if they are object Supervisors of both the queue and the target user.
- Request certain statistical information about any supervised object (such as the number of directories owned, the number of files owned, and the number of disk blocks used).
- Change the Supervisor-controlled field in files owned by supervised objects. These fields include the file attributes flags, the extended attributes (including read-audit and write-audit bits), the 64-byte extended directory information area, and the file owner (if the new target owner is also supervised by the caller).
- Change the owner of a directory if they supervise both the old directory owner and the proposed new directory owner.

5.4 Server-Based Bindery Functions

Function	Description
AddBinderyObjectToSet	Adds a bindery object to a property of type Set
ChangeBinderyObjectPassword	Changes the password of a bindery object
ChangeBinderyObjectSecurity	Changes the security of a bindery object
ChangePropertySecurity	Changes the security of a bindery objects property
CloseBindery	Closes the bindery
CreateBinderyObject	Creates a bindery object
CreateProperty	Creates a property for a bindery object
DeleteBinderyObject	Deletes a bindery object
DeleteBinderyObjectFromSet	Deletes a bindery object from a property of type Set
DeleteProperty	Deletes properties from a bindery object
GetBinderyAccessLevel	Returns the name and type of a bindery object
GetBinderyObjectID	Returns a bindery objects identification number
GetBinderyObjectName	Returns the name and type of a bindery object

Function	Description
IsBinderyObjectInSet	Determines if a bindery object is a member of a property of type Set
OpenBindery	Opens the bindery
ReadPropertyValue	Returns the value of a bindery objects property
RenameBinderyObject	Renames a bindery object
ScanBinderyObject	Scans the bindery for an object
ScanBinderyObjectTrusteePaths	Returns the paths to which an object has trustee rights
ScanProperty	Scans the bindery for an objects properties
VerifyBinderyObjectPassword	Verifies that the password of a bindery object is valid
WritePropertyValue	Writes a value to a property of type item

Server-Based Bindery Functions

6

This documentation alphabetically lists the Server-Based Bindery functions and describes their purpose, syntax, parameters, and return values.

AddBinderyObjectToSet

Adds a bindery object to a property of type Set (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWAddObjectToSet (page 28))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int AddBinderyObjectToSet (
   char   *objectName,
   WORD   objectType,
   char   *propertyName,
   char   *memberName,
   WORD   memberType);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object to which a new member is to be added (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object to which a new member is to be added (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

propertyName

(IN) Specifies the string containing the name of the property to which the member is to be added (maximum 16 characters, including the NULL terminator).

memberName

(IN) Specifies the string containing the name of the bindery object to be added to the property (maximum 48 characters, including the NULL terminator).

memberType

(IN) Specifies the type of the bindery object to be added (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

Return Values

0	(0x00)	ESUCCESS

150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
233	(0xE9)	ERR_MEMBER_ALREADY_EXISTS
235	(0xEB)	ERR_NOT_SET_PROPERTY
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
248	(0xF8)	ERR_NO_PROPERTY_WRITE_PRIVILEGE
251	(0xFB)	ERR_NO_SUCH_PROPERTY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName`, `objectType`, and `propertyName` parameters must uniquely identify a bindery object's property. These parameters must not contain wildcard characters. The property must be of type Set. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. The `propertyName` can be from 1 to 16 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `memberName` and `memberType` parameters must uniquely identify the bindery object to be added and must not contain wildcard characters.

This function searches consecutive segments of the property's value for an open slot where it can record the unique bindery object ID of the new member. The new member is inserted into the first available slot. If an open slot is not found, a new segment is created and the new member's unique bindery object ID is written to the first slot in the segment. The rest of the segment is filled with zeros.

This function requires write access to the property.

For example, to add user BILL's object ID to group STAFF's GROUP_MEMBERS Set property, a programmer passes BILL's object type (OT_USER) and object name (BILL) to the server with a call to AddBinderyObjectToSet. The server uses BILL's object type and object name to locate BILL's object ID. The server then searches all value segments associated with STAFF's GROUP_MEMBERS Set property until it finds an available slot. Finally, it inserts BILL's object ID number into the empty slot.

See Also

DeleteBinderyObjectFromSet (page 127), IsBinderyObjectInSet (page 139), WritePropertyValue (page 164)

AddBinderyObjectToSet Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>
```

```
main()
{
   int   completionCode;
   char  objectName[48], propertyName[16], memberName[48];
   WORD  objectType, memberType;

   /* Fill in the name of the object that is being added to */
   strcpy (objectName, "STAFF");

   /* Fill in the type of object that is being added to */
   objectType = OT_GROUP;

   /* Fill in the property name that is being added to */
   strcpy (propertyName, "GROUP_MEMBERS");

   /* Fill in the member name to be added */
   strcpy (memberName, "BILL");

   /* Fill in the member type to be added */
   memberType = OT_USER;
   completionCode = AddBinderyObjectToSet (objectName, objectType,
      propertyName, memberName, memberType);
   printf ("\n\n\n");
   if (completionCode)
      switch (completionCode)
      {
         case 150:
         printf ("SERVER OUT OF MEMORY\n");
         break;

         case 233:
         printf ("MEMBER ALREADY EXISTS\n");
         break;

         case 235:
         printf ("NOT GROUP PROPERTY\n");
         break;

         case 240:
         printf ("WILDCARD NOT ALLOWED\n");
         break;

         case 248:
         printf ("NO PROPERTY WRITE PRIVILEGE\n");
         break;

         case 251:
         printf ("NO SUCH PROPERTY\n");
         break;

         case 252:
         printf ("NO SUCH OBJECT\n");
         break;
```

```
            case 254:
            printf ("SERVER BINDERY LOCKED\n");
            break;

            case 255:
            printf ("BINDERY FAILURE\n");
            break;

            case default:
            printf ("completionCode = %d\n", completionCode);
            break;
         }
      else
         printf (" SUCCESSFULLY added member %s\n", memberName);
}
```

ChangeBinderyObjectPassword

Changes the password of a bindery object (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWChangeObjectPassword (page 31))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int ChangeBinderyObjectPassword (
   char   *objectName,
   WORD   objectType,
   char   *oldPassword,
   char   *newPassword);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

oldPassword

(IN) Specifies the string containing the current password to be checked for validity (maximum 128 characters; NULL string = no password).

newPassword

(IN) Specifies the string containing the new password to be assigned to the bindery object (maximum 128 characters; NULL string = no password).

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
251	(0xFB)	ERR_NO_SUCH_PROPERTY

252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_NO_SUCH_OBJECT_OR_BAD_PASSWORD

Remarks

The `objectName` and `objectType` parameters must uniquely identify the bindery object and must not contain any wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

This function creates a property PASSWORD if the bindery object does not have one. It also assigns the property security (0x44) to the property PASSWORD. (Only the NetWare® OS can find the property or add value to the property.) The PASSWORD property is created with an associated bindery read and write access level, and the password property value is assigned the `newPassword`. A password can be from 1 to 128 characters long, including the NULL terminator.

Passwords are case sensitive. You can create passwords in lowercase or uppercase characters. When logging in, however, they must be entered exactly as when they were created.

There is a distinction between a bindery object without a password property and a bindery object with a password property that has no value. An entity is not allowed to log in to a server as a bindery object that does not have a password property. However, an entity is allowed to log in to a server as a bindery object that has a password property with no value. This function does not require that the requesting entity be logged in to the server.

This function requires read and write access to the bindery object.

See Also

VerifyBinderyObjectPassword (page 162)

ChangeBinderyObjectSecurity

Changes the security of a bindery object (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWChangeObjectSecurity (page 33))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int ChangeBinderyObjectSecurity (
   char   *objectName,
   WORD   objectType,
   BYTE   newObjectSecurity);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

newObjectSecurity

(IN) Indicates the read/write access to the bindery object.

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
241	(0xF1)	ERR_INVALID_BINDERY_SECURITY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName` and `objectType` parameters must uniquely identify the bindery object and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `newObjectSecurity` parameter is actually two nibbles. The low-order nibble determines who can scan for and find the object. The high-order nibble determines who can add properties to the object. The following values are defined for each nibble:

0	0 0 0 0	Anyone
1	0 0 0 1	Logged
2	0 0 1 0	Object
3	0 0 1 1	Supervisor
4	0 1 0 0	NetWare Operating System

For example, 0x31 indicates that any user logged in to the server can find the object, but only the supervisor can add a property to the object.

See Also

ChangePropertySecurity (page 113)

ChangeBinderyObjectSecurity Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
   int    completionCode;
   char   objName[48];
   WORD   objType;
   BYTE   newObjSecurity;
   printf ("\n\n");
   printf ("Enter Name of Object -> ");
   scanf ("%s", objName);
   printf ("\nEnter Specifies the type of Object -> ");
   scanf ("%d", &objType);
   printf ("\nEnter the new READ/WRITE access -> ");
   scanf ("%2x", &newObjSecurity);
   completionCode = ChangeBinderyObjectSecurity (objName,
      objType, newObjSecurity );
   if (completionCode)
   {
      printf ("\n\n\n");
      if (completionCode == 150)
         printf ("SERVER OUT OF MEMORY\n\n");
      if (completionCode == 240)
```

```
            printf ("WILDCARD NOT ALLOWED\n\n");
         if (completionCode == 241)
            printf ("INVALID BINDERY SECURITY\n\n");
         if (completionCode == 251)
            printf ("NO SUCH PROPERTY\n\n");
         if (completionCode == 252)
            printf ("NO SUCH OBJECT\n\n");
         if (completionCode == 254)
            printf ("SERVER BINDERY LOCKED\n\n");
         if (completionCode == 255)
            printf ("BINDERY FAILURE\n\n");
      }
      else if (completionCode == 0)
         printf ("\n\n\nObject Security of %s is %d\n", objName,
            newObjSecurity);
      else
         printf ("completionCode = %d\n", completionCode);
   }
```

ChangePropertySecurity

Changes the security of a bindery object's property (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWChangePropertySecurity (page 35))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int ChangePropertySecurity (
   char   *objectName,
   WORD   objectType,
   char   *propertyName,
   BYTE   newPropertySecurity);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

propertyName

(IN) Specifies the string containing the name of the property whose security is to be changed (maximum16 characters, including the NULL terminator).

newPropertySecurity

(IN) Indicates the read/write access of others to the property.

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
241	(0xF1)	ERR_INVALID_BINDERY_SECURITY
251	(0xFB)	ERR_NO_SUCH_PROPERTY

252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName`, `objectType`, and `propertyName` parameters must uniquely identify the property and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. The `propertyName` can be from 1 to 15 characters long. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `newPropertySecurity` parameter is actually two nibbles. The low-order nibble determines who can scan for and find the object. The high-order nibble determines who can add properties to the object. The following values are defined for each nibble:

0	0 0 0 0	Anyone
1	0 0 0 1	Logged
2	0 0 1 0	Object
3	0 0 1 1	Supervisor
4	0 1 0 0	NetWare Operating System

For example, 0x31 indicates that any user logged in to the server can find the object, but only the supervisor can add a property to the object.

See Also

ChangeBinderyObjectSecurity (page 110)

ChangePropertySecurity Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
   int   completionCode;
   char  objectName[48];
   int   objectType;
   char  propertyName[16];
   BYTE  newAccessFlags;
   strcpy (objectName, "JDOE");
   objectType = OT_USER;
   strcpy (propertyName, "GROUP_MEMBERS");
   newAccessFlags = 0x31;
   completionCode =ChangePropertySecurity (objectName, objectType,
      propertyName, newAccessFlags);
```

```
   if (completionCode)
      switch (completionCode)
      {
         case 150:
         printf ("SERVER OUT OF MEMORY\n\n");
         break;

         case 240:
         printf ("WILDCARD NOT ALLOWED\n\n");
         break;

         case 241:
         printf ("INVALID BINDERY SECURITY\n\n");
         break;

         case 251:
         printf ("NO SUCH PROPERTY\n\n");
         break;

         case 252:
         printf ("NO SUCH OBJECT\n\n");
         break;

         case 254:
         printf ("SERVER BINDERY LOCKED\n\n");
         break;

         case 255:
         printf ("BINDERY FAILURE\n\n");
         break;

         case default:
         printf ("completionCode=%d\n", completionCode);
         break;
      }
   else
      printf ("Access flags for %s is %2X\n",
      propertyName,
      newAccessFlags);
}
```

CloseBindery

Closes the bindery (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWCloseBindery (page 38))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int CloseBindery (void);
```

Return Values

0	(0x00)	ESUCCESS
	(0xFF)	FAILURE

Remarks

The bindery files are normally kept open and locked so that they cannot be directly accessed. bindery files need to be closed when archiving or restoring.

While the bindery is closed, most functions of the network are disabled. Therefore, the time that the bindery is closed should be kept to a minimum.

Only the supervisor or a bindery object that is security-equivalent to the supervisor can close the bindery.

If an application closes the bindery, an open bindery needs to follow before an end-of-job (EOJ) occurs. This is because most functions of the network are disabled after a bindery is closed and that can include being able to run another application to reopen the bindery.

See Also

OpenBindery (page 142)

CloseBindery Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
```

```
{
   int  completionCode;
/*
   NOTE: When the bindery is closed, most network functions
   are disabled. Use the CloseBindery function with care.
   Also, be sure to call OpenBindery before an endofjob
   (EOJ) occurs.
*/
   completionCode = CloseBindery ();
   printf ("completionCode = %d\n", completionCode);
/*
   At this point, perform the steps you need (such as backing
   up a networked system); then reopen the bindery before an
   endofjob (EOJ) occurs.
*/
   completionCode = OpenBindery ();
  printf ("completionCode from Open bindery = %d\n", completionCode);
}
```

CreateBinderyObject

Creates a bindery object (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWCreateObject (page 40))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int CreateBinderyObject (
   char   *objectName,
   WORD   objectType,
   BYTE   objectFlag,
   BYTE   objectSecurity);
```

Parameters

objectName

(IN) Specifies the string containing the name of the new bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the new bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

objectFlag

(IN) Indicates whether the new bindery object is Dynamic or Static (BF_DYNAMIC or BF_STATIC).

objectSecurity

(IN) Indicates the read/write access of others to the new bindery object.

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
238	(0xEE)	ERR_OBJECT_ALREADY_EXISTS
239	(0xEF)	ERR_INVALID_NAME
241	(0xF1)	ERR_INVALID_BINDERY_SECURITY

245	(0xF5)	ERR_NO_OBJECT_CREATE_PRIVILEGE
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName` and `objectType` parameters must uniquely identify the bindery object and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `objectFlag` parameter is a one-byte parameter which indicates whether the object is Dynamic (0x01) or Static (0x00). A dynamic object is an object that is created and deleted frequently. Dynamic objects are deleted from the bindery when the server is initialized or when the object is specifically deleted. Static objects remain in the bindery until deleted with DeleteBinderyObject.

The `objectSecurity` parameter is actually two nibbles. The low-order nibble determines who can scan for and find the object. The high-order nibble determines who can add properties to the object. The following values are defined for each nibble:

0	0 0 0 0	Anyone
1	0 0 0 1	Logged
2	0 0 1 0	Object
3	0 0 1 1	Supervisor
4	0 1 0 0	NetWare Operating System

For example, 0x31 indicates that any user logged in to the server can find the object, but only the supervisor can add a property to the object.

The bindery object must have a password property to log in to a server. The ChangeBinderyObjectPassword and CreateProperty functions add the property PASSWORD to an object.

See Also

ChangeBinderyObjectPassword (page 108), CreateProperty (page 121), DeleteBinderyObject (page 125), RenameBinderyObject (page 148), ScanBinderyObject (page 151)

CreateBinderyObject Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
   int    completionCode;
   char   objectName[48];
```

```
   WORD   ;
   BYTE   objectFlag;
   BYTE   objectSecurity;
   strcpy (objectName, "JDOE");
   objectType = 1;
   objectFlag = BF_STATIC;
   objectSecurity = 0x31;
   completionCode = CreateBinderyObject (objectName, objectType,
      objectFlag, objectSecurity);
   if (completionCode == 0)
      printf ("Successfully created object %s of type %d\n",
               objectName, objectType);
   else
      printf ("Error %d in CreateBinderyObject\n",
      completionCode);
}
```

CreateProperty

Creates a property for a bindery object (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWCreateProperty (page 43))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int CreateProperty (
   char   *objectName,
   WORD   objectType,
   char   *propertyName,
   BYTE   propertyFlags,
   BYTE   propertySecurity);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

propertyName

(IN) Specifies the string containing the name of the property to be created (maximum 16 characters, including the NULL terminator).

propertyFlags

(IN) Indicates whether the new property is Dynamic or Static and whether it is of type Item or Set: BF_DYNAMIC or BF_STATIC is logically ORed with BF_ITEM or BF_SET (BF_STATIC | BF_ITEM, and so on).

propertySecurity

(IN) Indicates the read/write access of others to the new property.

Return Values

0	(0x00)	ESUCCESS

150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
237	(0xED)	ERR_PROPERTY_ALREADY_EXISTS
239	(0xEF)	ERR_INVALID_NAME
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
241	(0xF1)	ERR_INVALID_BINDERY_SECURITY
247	(0xF7)	ERR_NO_PROPERTY_CREATE_PRIVILEGE
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName` and `objectType` parameters must uniquely identify the bindery object and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. The `propertyName` can be from 1 to 16 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `propertyFlags` parameter indicates whether the property is Dynamic or Static and whether it is of type Set or Item. A dynamic property is created and deleted frequently. Dynamic properties are deleted from the bindery when the server is initialized. Static properties remain in the bindery until deleted with DeleteProperty.

The property type indicates the type of data stored in the value of a property. The value of a Set property contains a series of bindery object IDs. Each object ID is 4 bytes long. The value of an Item property contains segments of 128-byte strings.

The `propertySecurity` parameter is actually two nibbles. The low-order nibble determines who can scan for and find the object. The high-order nibble determines who can add properties to the object. The following values are defined for each nibble:

0	0 0 0 0	Anyone
1	0 0 0 1	Logged
2	0 0 1 0	Object
3	0 0 1 1	Supervisor
4	0 1 0 0	NetWare Operating System

For example, 0x31 indicates that any user logged in to the server can find the property, but only the supervisor can add a value to the property.

The requesting workstation cannot create properties that have security greater than the workstation's access to the bindery object. The password property can also be created with ChangeBinderyObjectPassword. CreateProperty can also be used to create the password property.

See Also

ChangeBinderyObjectPassword (page 108), DeleteProperty (page 130)

CreateProperty Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
   int   completionCode;
   char  objectName[48];
   char  propertyName[16];
   WORD  objectType;
   BYTE  propertyFlags;
   BYTE  propertySecurity;

   strcpy (objectName, "BRUTH");
   objectType = OT_USER;
   strcpy (propertyName, "BASEBALL_TEAM");
   propertyFlags = BF_STATIC | BF_SET;
   propertySecurity = 0x31;
   completionCode = CreateProperty (objectName, objectType,
      propertyName, propertyFlags, propertySecurity);
   if (completionCode)
      switch (completionCode)
      {
      case 150:
      printf ("SERVER OUT OF MEMORY\n");
      break;

      case 237:
      printf ("PROPERTY ALREADY EXISTS\n");
      break;

      case 239:
      printf ("INVALID NAME\n");
      break;

      case 240:
      printf ("WILDCARD NOT ALLOWED\n");
      break;

      case 241:
      printf ("INVALID BINDERY SECURITY\n");
      break;

      case 247:
      printf ("NO PROPERTY CREATE PRIVILEGE\n");
      break;

      case 252:
```

```
        printf ("NO SUCH OBJECT\n");
        break;

        case 254:
        printf ("SERVER BINDERY LOCKED\n");
        break;

        case 255:
        printf ("BINDERY FAILURE\n");
        break;

        case default:
        printf ("completionCode = %d\n", completionCode);
        break;
        }
        else
        printf ("SUCCESSFULLY created property %s\n",
        propertyName);
}
```

DeleteBinderyObject

Deletes a bindery object (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWDeleteObject (page 46))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int DeleteBinderyObject (
   char   *objectName,
   WORD   objectType);
```

Parameters

`objectName`

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

`objectType`

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
239	(0xEF)	ERR_INVALID_NAME
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
244	(0xF4)	ERR_NO_OBJECT_DELETE_PRIVILEGE
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName` and `objectType` parameters must uniquely identify the bindery object to be deleted and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

See Also

CreateBinderyObject (page 118), RenameBinderyObject (page 148)

DeleteBinderyObject Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
   int    completionCode;
   char   objectName[48];
   WORD   objectType;

   strcpy (objectName, "PROSE");
   objectType = OT_USER;
   completionCode = DeleteBinderyObject (objectName, objectType);
   if (completionCode == 0)
      printf ("%s  of Type  %d has been deleted\n", objectName,
objectType);
   else
      printf ("Error %d in DeleteBinderyObject\n", completionCode);
}
```

DeleteBinderyObjectFromSet

Deletes a bindery object from a property of type Set (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWDeleteObjectFromSet (page 48))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int DeleteBinderyObjectFromSet (
   char   *objectName,
   WORD   objectType,
   char   *propertyName,
   char   *memberName,
   WORD   memberType);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object from which the member is to be deleted (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

propertyName

(IN) Specifies the string containing the name of a set property (maximum 16 characters, including the NULL terminator).

memberName

(IN) Specifies the string containing the name of the bindery object to be deleted from the set (maximum 48 characters, including the NULL terminator).

memberType

(IN) Specifies the type of bindery object to be deleted from the set (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

Return Values

0	(0x00)	ESUCCESS

150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
234	(0xEA)	ERR_NO_SUCH_MEMBER
235	(0xEB)	ERR_NOT_GROUP_PROPERTY
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
248	(0xF8)	ERR_NO_PROPERTY_WRITE_PRIVILEGE
251	(0xFB)	ERR_NO_SUCH_PROPERTY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName`, `objectType`, and `propertyName` parameters must uniquely identify a bindery object's properties. These parameters must not contain wildcard characters. The property must be of type Set. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. The `propertyName` can be from 1 to 16 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `memberName` and `memberType` parameters must uniquely identify the bindery object to be deleted and must not contain wildcard characters.

This function searches consecutive segments of the property's value for a unique object ID that matches the unique object ID of the member to be deleted. When the member is found it is deleted. The remaining IDs in the segment are shifted and the last previously used slot in the segment is filled with zeros. This ensures that IDs within a segment are packed. However, IDs are not packed between segments.

See Also

AddBinderyObjectToSet (page 104), IsBinderyObjectInSet (page 139)

DeleteBinderyObjectFromSet Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
int   completionCode;
char  objectName[48];
char  propertyName[16];
char  memberName[48];
WORD  objectType, memberType;
      strcpy (objectName, "SUPERVISOR");
      objectType = OT_USER;
      strcpy (propertyName, "BASEBALL_TEAM");
```

```
        strcpy (memberName, "PROSE");
        memberType = OT_USER;
        if (completionCode)
        switch (completionCode)
        {
        case 150:
        printf ("SERVER OUT OF MEMORY\n");
        break;

        case 234:
        printf ("NO SUCH MEMBER\n");
        break;

        case 235:
        printf ("NOT GROUP PROPERTY\n");
        break;

        case 240:
        printf ("WILDCARD NOT ALLOWED\n");
        break;

        case 248:
        printf ("NO PROPERTY WRITE PRIVILEGE\n");
        break;

        case 251:
        printf ("NO SUCH PROPERTY\n");
        break;

        case 252:
        printf ("NO SUCH OBJECT\n");
        break;

        case 254:
        printf ("SERVER BINDERY LOCKED\n");
        break;

        case 255:
        printf ("BINDERY FAILURE\n");
        break;

        case default:
        printf ("completionCode = %d\n",
        completionCode);
        break;
        }
        else
        printf ("SUCCESSFULLY deleted member %s from %s\n",
        memberName, propertyName);
}
```

DeleteProperty

Deletes properties from a bindery object (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWDeleteProperty (page 50))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int DeleteProperty (
   char   *objectName,
   WORD   objectType,
   char   *propertyName);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object from which the property is to be deleted (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

propertyName

(IN) Specifies the string containing the name of the property to be deleted (maximum 16 characters; can contain wildcard characters, including the NULL terminator).

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
241	(0xF1)	ERR_INVALID_BINDERY_SECURITY
246	(0xF6)	ERR_NO_PROPERTY_DELETE_PRIVILEGE
251	(0xFB)	ERR_NO_SUCH_PROPERTY
252	(0xFC)	ERR_NO_SUCH_OBJECT

254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName` and `objectType` parameters must uniquely identify the bindery object and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `propertyName` parameter contains the name of the property to be deleted and can be from 1 to 16 characters long, including the NULL terminator. Wildcard characters are allowed, but slashes, backslashes, commas, colons, semicolons, asterisks, and question marks are prohibited. All matching properties of the bindery object are deleted when the `propertyName` parameter contains wildcard characters.

See Also

CreateProperty (page 121)

DeleteProperty Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
int   completionCode;
char  objectName[48];
char  propertyName[16];
WORD  objectType;

   strcpy (objectName, "PROSE");
   objectType = OT_USER;
   strcpy (propertyName, "BASEBALL_TEAM");
   completionCode = DeleteProperty (objectName, objectType,
      propertyName);
   if (completionCode)
   switch (completionCode)
   {
      case 150:
      printf ("SERVER OUT OF MEMORY\n");

      case 240:
      printf ("WILDCARD NOT ALLOWED\n");

      case 241:
      printf ("INVALID BINDERY SECURITY\n");

      case 246:
      printf ("NO PROPERTY DELETE PRIVILEGE\n");
```

```
            case 251:
            printf ("NO SUCH PROPERTY\n");

            case 252:
            printf ("NO SUCH OBJECT\n");

            case 254:
            printf ("SERVER BINDERY LOCKED\n");

            case 255:
            printf ("BINDERY FAILURE\n");

            case default:
            printf ("Error %d in DeleteProperty\n",
            completionCode);
         }
         else
            printf ("SUCCESSFULLY deleted %s from %s's properties\n",
                     propertyName,
                     objectName);
}
```

GetBinderyAccessLevel

Indicates the access level of the current connection number to the server's bindery (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetBinderyAccessLevel (page 54))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int GetBinderyAccessLevel (
   BYTE   *securityAccessLevel,
   long   *objectID);
```

Parameters

securityAccessLevel

(OUT) Receives the security access level the requesting object has to the bindery.

objectID

(OUT) Receives the unique ID of the bindery object logged in to the current connection or else a value of -1 if no object is logged in on the current connection.

Return Values

0	(0x00)	ESUCCESS

Remarks

Object security determines who can access the object. The low-order nibble determines who can read (scan for and find) the object. The high-order nibble determines who can write to (add properties to or delete properties from) the object.

The following values are defined for each nibble:

0	0 0 0 0	Anyone
1	0 0 0 1	Logged
2	0 0 1 0	Object
3	0 0 1 1	Supervisor

For example, 0x31 indicates that any user logged in to the server can find the object, but only the supervisor can add a property to the object.

See Also

GetBinderyObjectName (page 137), ScanBinderyObject (page 151), ScanProperty (page 158)

GetBinderyAccessLevel Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
int    completionCode;
BYTE   securityAccessLevel;
long   objectID;

   GetBinderyObjectID ("JDOE", OT_USER, &objectID);
   completionCode = GetBinderyAccessLevel (&securityAccessLevel,
   objectID);
   if (completionCode == 0)
   {
      printf (" \n\n\nSUCCESSFUL\n\n");
      printf ("Security Access Level -> %2X\n",
      securityAccessLevel);
   }
   else
      printf ("Error %d in GetBinderyAccessLevel\n",
               completionCode);
}
```

GetBinderyObjectID

Returns a bindery object's unique identification number (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetObjectID (page 58))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int GetBinderyObjectID (
   char   *objectName,
   WORD   objectType,
   long   *objectID);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

objectID

(OUT) Receives the unique bindery object ID number.

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
239	(0xEF)	ERR_INVALID_NAME
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName` and `objectType` parameters must uniquely identify the bindery object and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

See Also

GetBinderyObjectName (page 137)

GetBinderyObjectID Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
char  objName[48];
WORD  objType;
long  objectID;
int   completionCode;

  printf ("Enter Object Name:      ");
  scanf ("%s", objName);
  printf ("\nEnter Object Type:    ");
  scanf ("%d", &objType);
  completionCode = GetBinderyObjectID (objName, objType,
      &objectID);
  if (completionCode)
  {
      if (completionCode == 150)
         printf ("SERVER OUT OF MEMORY\n\n");
      if (completionCode == 239)
         printf ("INVALID NAME\n\n");
      if (completionCode == 240)
         printf ("WILDCARD NOT ALLOWED\n\n");
      if (completionCode == 252)
         printf ("NO SUCH OBJECT\n\n");
      if (completionCode == 254)
         printf ("SERVER BINDERY LOCKED\n\n");
      if (completionCode == 255)
         printf ("BINDERY FAILURE\n\n");
  }
  else
     printf ("\n\nObject ID for %s is...     %8lX\n",
              objName, objectID);
}
```

GetBinderyObjectName

Returns the name and type of a bindery object (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetObjectName (page 65))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int GetBinderyObjectName (
   long   objectID,
   char   *objectName,
   WORD   *objectType);
```

Parameters

objectID

(IN) Specifies a unique bindery object ID.

objectName

(OUT) Receives a string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(OUT) Receives the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The value in the `objectID` parameter is a 4-byte number that identifies a bindery object. It is assigned by the operating system.

The `objectName` and `objectType` parameters uniquely identify the bindery object and do not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters are used.

See Also

GetBinderyObjectID (page 135)

GetBinderyObjectName Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
char  objectName[48];
WORD  objectType;
long  objectID;
int   completionCode;

   printf ("Enter Object ID:    ");
   scanf ("%8lX", &objectID);
   completionCode = GetBinderyObjectName (objectID, objectName,
                    &objectType);
   if (completionCode)
   {
       if (completionCode == 150)
          printf ("SERVER OUT OF MEMORY\n\n");
       if (completionCode == 252)
          printf ("NO SUCH OBJECT\n\n");
       if (completionCode == 254)
          printf ("SERVER BINDERY LOCKED\n\n");
       if (completionCode == 255)
          printf ("BINDERY FAILURE\n\n");
       else
          printf ("Error %d in GetBinderyObjectName\n",
                   completionCode);
   }
   else
   {
       printf ("\n\n\nObject Name...      %s\n", objectName);
       printf ("Object Type...      %d\n", objectType);
   }
}
```

IsBinderyObjectInSet

Determines whether a bindery object is a member of a property of type Set (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWIsObjectInSet (page 67))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int IsBinderyObjectInSet (
   char   *objectName,
   WORD   objectType,
   char   *propertyName,
   char   *memberName,
   WORD   memberType);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

propertyName

(IN) Specifies the string containing the name of the Set property (maximum 16 characters, including the NULL terminator).

memberName

(IN) Specifies the string containing the bindery object name to be checked for set membership (maximum 48 characters, including the NULL terminator).

memberType

(IN) Specifies a member's bindery object type (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

Return Values

0	(0x00)	ESUCCESS

150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
234	(0xEA)	ERR_NO_SUCH_MEMBER
235	(0xEB)	ERR_NOT_GROUP_PROPERTY
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
249	(0xF9)	ERR_NO_PROPERTY_READ_PRIVILEGE
251	(0xFB)	ERR_NO_SUCH_PROPERTY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName`, `objectType`, and `propertyName` parameters must uniquely identify a bindery object's property and must not contain wildcard characters. The property must be of type Set. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. The `propertyName` can be from 1 to 16 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `memberName` and `memberType` parameters must uniquely identify the bindery object to be searched and must not contain wildcard characters.

IsBinderyObjectInSet scans segments of a Set property for the object ID of the specified `memberName`.

See Also

AddBinderyObjectToSet (page 104), DeleteBinderyObjectFromSet (page 127)

IsBinderyObjectInSet Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
int   completionCode;
char  objectName[48];
char  propertyName[16];
char  memberName[48];
WORD  objectType, memberType;

   strcpy (objectName, "SUPERVISOR");
   objectType = OT_USER;
   strcpy (propertyName, "BASEBALL_TEAM");
   strcpy (memberName, "PROSE");
   memberType = OT_USER;
```

```
   completionCode =  IsBinderyObjectInSet (objectName, objectType,
       propertyName, memberName, memberType);
   if (completionCode)
   switch (completionCode)
   {
      case 150:
      printf ("SERVER OUT OF MEMORY\n");
      break;

      case 234:
      printf ("NO SUCH MEMBER\n");
      break;

      case 235:
      printf ("NOT GROUP PROPERTY\n");
      break;

      case 240:
      printf ("WILDCARD NOT ALLOWED\n");
      break;

      case 248:
      printf ("NO PROPERTY WRITE PRIVILEGE\n");
      break;

      case 249:
      printf ("NO PROPERTY READ PRIVILEGE\n");
      break;

      case 251:
      printf ("NO SUCH PROPERTY\n");
      break;

      case 252:
      printf ("NO SUCH OBJECT\n");
      break;

      case 254:
      printf ("SERVER BINDERY LOCKED\n");
      break;

      case 255:
      printf ("BINDERY FAILURE\n");
      break;

      case default:
      printf ("completionCode = %d\n", completionCode);
      break;
   }
   else
   printf (" SUCCESSFULLY COMPLETED\n ");
}
```

OpenBindery

Opens the bindery (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWOpenBindery (page 70))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int OpenBindery (void);
```

Return Values

0	(0x00)	ESUCCESS

Remarks

The bindery files are normally kept open and locked. Therefore, this function is only required after CloseBindery is called. No other bindery calls can be serviced while the bindery is closed.

See Also

CloseBindery (page 116)

OpenBindery Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
int  completionCode;

/*  NOTE: When the bindery is closed, most network functions
    are disabled. Use the CloseBindery function with care.
    Also, be sure to call OpenBindery before an endofjob
    (EOJ) occurs.
*/

   completionCode = CloseBindery ();
   printf ("completionCode = %d\n", completionCode);
```

```
/*  At this point, perform the steps you need (such as backing
    up a networked system); then reopen the bindery before an
    endofjob (EOJ) occurs.
*/

   completionCode = OpenBindery ();
   printf ("completionCode from Open Bindery = %d\n",
   completionCode);
}
```

ReadPropertyValue

Returns the value of a bindery object's property (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWReadPropertyValue (page 72))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int ReadPropertyValue (
   char   *objectName,
   WORD   objectType,
   char   *propertyName,
   int    segmentNumber,
   BYTE   *propertyValue,
   BYTE   *moreSegments,
   BYTE   *propertyFlags);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

propertyName

(IN) Specifies the string containing the name of the property (maximum 16 characters, including the NULL terminator).

segmentNumber

(IN) Specifies the number of the data to be read: 1 = First segment of the property's value.

propertyValue

(OUT) Receives a buffer containing a segment of the property's value (128 bytes).

moreSegments

(OUT) Receives a flag that indicates if the property value has more data segments after the current segment (0 = no more segments, 255 = more segments follow).

propertyFlags

(OUT) Receives the property flags of the property: BF_DYNAMIC or BF_STATIC is logically ORed with BF_ITEM or BF_SET (BF_STATIC | BF_ITEM, and so on).

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
236	(0xEC)	ERR_NO_SUCH_SEGMENT
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
241	(0xF1)	ERR_INVALID_BINDERY_SECURITY
249	(0xF9)	ERR_NO_PROPERTY_READ_PRIVILEGE
251	(0xFB)	ERR_NO_SUCH_PROPERTY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

This function passes the `objectName`, `objectType`, `propertyName`, and `segmentNumber` parameters. It returns the value of a bindery object's property through the `propertyValue` parameter. The `moreSegments` parameter returns a flag indicating whether more segments exist, and the `propertyFlags` parameter returns the property flags of the property.

The `objectName`, `objectType`, and `propertyName` parameters must uniquely identify a bindery object's property and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. The `propertyName` can be from 1 to 16 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `propertyFlags` parameter indicates whether the property is Dynamic or Static and whether the property is of type Set or Item. A Dynamic property is a property that is created and deleted frequently. Dynamic properties are deleted from the bindery when the server is initialized. Static properties remain in the bindery until deleted with DeleteProperty.

The property type indicates the type of data stored in the value of a property. The value of a Set property contains a series of bindery object IDs. Each object ID is 4 bytes long. The value of an Item property contains segments of 128-byte strings.

Bit 1 of the `propertyFlags` parameter are set as follows:

Figure 6-1 *The propertyFlags Parameter*

7	6	5	4	3	2	1	0	
							0	The property is Static.
							1	The property is Dynamic.
						0		The property is an Item.
						1		The property is a Set.

For example, 0x2 indicates that a property is Static and of type Set.

This function is used iteratively to read property values with more than 128 bytes of data. The segment number should be set to 1 to read the first data segment of a property and then incremented for each subsequent call until the `moreSegments` flag is set to 0 or NO_SUCH_SEGMENT is returned.

The data returned in the `propertyValue` parameter is an array of bindery object IDs, if the property is of type Set. When reading the value of a property of type Set, a member's bindery object ID of 0 indicates the end of a segment. When a member is deleted from a set, compaction occurs only within its segment. Therefore, to ensure that all members of a set are read, continue reading segments until the bindery sets the `moreSegments` flag to 0 or returns a NO_SUCH_SEGMENT error.

The bindery makes no attempt to coordinate activities between multiple entities that might be reading or writing data to a single property. This means that one entity might read a partially updated property and get inconsistent data if the property's data extends across multiple segments. If this presents a problem, coordination on reads and writes must be handled by application programs. Logical record locks can be used to coordinate activities among applications.

See Also

GetBinderyObjectName (page 137), WritePropertyValue (page 164)

ReadPropertyValue Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

/*
 *  This example gets the network and node address of a file
 *  server from the NET_ADDRESS property.
 */

main()
{
BYTE netaddr[128], more, flags;

   printf( "return code = %d\r\n",
      ReadPropertyValue( "ROSS", 4, "NET_ADDRESS", 1, netaddr,
```

```
      &more, &flags));
   printf(
      "%02X%02X%02X%02X%02X%02X%02X%02X%02X%02Xmore=%x,flags=%x\r\n,
      netaddr[0], netaddr[1], netaddr[2], netaddr[3],
      netaddr[4], netaddr[5], netaddr[6], netaddr[7], netaddr[8],
      netaddr[9], more, flags );
}
```

RenameBinderyObject

Renames a bindery object (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWRenameObject (page 75))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int RenameBinderyObject (
   char   *objectName,
   char   *newObjectName,
   WORD   objectType);
```

Parameters

objectName

(IN) Specifies the string containing the name of a currently defined bindery object (maximum 48 characters, including the NULL terminator).

newObjectName

(IN) Specifies the string containing the new name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
239	(0xEF)	ERR_INVALID_NAME
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
243	(0xF3)	ERR_NO_OBJECT_RENAME_PRIVILEGE
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED

255	(0xFF)	ERR_BINDERY_FAILURE

Remark

The `objectName` and `objectType` parameters must uniquely identify the bindery object and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

See Also

CreateBinderyObject (page 118), DeleteBinderyObject (page 125)

RenameBinderyObject Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
int   completionCode;
char  objectName[48];
char  newObjectName[48];
WORD  objectType;

   printf ("\n\n  -- RENAME BINDERY OBJECT --\n\n ");
   printf ("FROM:  ");
   scanf ("%s", objectName);
   printf ("\n TO:    ");
   scanf ("%s", newObjectName);
   printf ("\nEnter type of object:  ");
   scanf ("%d", &objectType);
   completionCode = RenameBinderyObject (objectName,
       newObjectName, objectType);
   if (completionCode)
   {
      printf ("\n\n\n");
      if (completionCode == 150)
         printf ("SERVER OUT OF MEMORY\n\n");
      if (completionCode == 239)
         printf ("INVALID NAME\n\n");
      if (completionCode == 240)
         printf ("WILDCARD NOT ALLOWED\n\n");
      if (completionCode == 252)
         printf ("NO SUCH OBJECT\n\n");
      if (completionCode == 254)
         printf ("SERVER BINDERY LOCKED\n\n");
      if (completionCode == 255)
         printf ("BINDERY FAILURE\n\n");
   }
   else
   printf ("\n\n\nSUCCESSFULLY renamed %s to %s\n",
```

```
                    objectName, newObjectName);
        }
```

ScanBinderyObject

Scans the bindery for an object (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWScanObject (page 77))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int ScanBinderyObject (
   char   *searchObjectName,
   WORD   searchObjectType,
   long   *objectID,
   char   *objectName,
   WORD   *objectType,
   char   *objectHasProperties,
   char   *objectFlag,
   char   *objectSecurity);
```

Parameters

searchObjectName

(IN) Specifies the string containing the bindery object to search for (maximum 48 characters; can contain wildcard characters, including the NULL terminator).

searchObjectType

(IN/OUT) Specifies the type of the bindery object to search for (OT_WILD, OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

objectID

(IN/OUT) Contains the object ID from the previous search (initial search requires a -1), and receives the unique bindery object ID for the matching object.

objectName

(OUT) Receives a NULL-terminated string containing the name of the matching bindery object (maximum 48 characters, including the NULL terminator).

objectType

(OUT) Receives the type of the matching bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

objectHasProperties

(OUT) Receives a flag that indicates if the bindery object has properties to scan: 0 = No properties for object. 255 = Object has properties.

objectFlag

(OUT) Receives a flag that indicates if the matching bindery object is Dynamic or Static (BF_DYNAMIC or BF_STATIC).

objectSecurity

(OUT) Receives a flag that indicates the read and write access of others to the matching bindery object.

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
239	(0xEF)	ERR_INVALID_NAME
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

This function is used iteratively to scan the bindery for all objects that match both the `searchObjectName` and the `searchObjectType` parameters.

The `objectID` parameter should be set to -1 for the first search. Upon return, the `objectID` parameter receives a number which is the value in the `objectID` parameter for the next call. This function can scan for one particular object type or all object types (WILD). It can also scan for a specific object name, or it can use wildcard characters to scan for a group of related object names.

The `objectSecurity` parameter is actually two nibbles. The low-order nibble determines who can scan for and find the object. The high-order nibble determines who can add properties to the object. The following values are defined for each nibble:

0	0 0 0 0	Anyone
1	0 0 0 1	Logged
2	0 0 1 0	Object
3	0 0 1 1	Supervisor
4	0 1 0 0	NetWare Operating System

For example, 0x31 indicates that any user logged in to the server can find the object, but only the supervisor can add a property to the object.

See Also

ChangeBinderyObjectSecurity (page 110), CreateBinderyObject (page 118), DeleteBinderyObject (page 125), ScanProperty (page 158)

ScanBinderyObject Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
int   completionCode;
char  searchObjectName[48];
WORD  searchObjectType;
long  objectID = -1;
char  objectName[48];
WORD  objectType;
char  objectHasProperties;
char  objectFlag;
char  objectSecurity;

   printf ("\n\n");
   printf ("Enter Object name to search:  ");
   scanf ("%s", searchObjectName);
   printf ("\nEnter object type to search:  ");
   scanf ("%4X", &searchObjectType);
   printf ("\nObject name...     %s\n", searchObjectName);
   printf ("\nObject type...     %d\n", searchObjectType);
   printf ("\nObject type...     %4X\n", searchObjectType);
   printf ("\n\n\n\n\n\n");
   printf ("OBJECT NAME        OBJECT TYPE       OBJECT ID\n");
   printf ("————-        ———-       ——\n\n");
   for (objectID = -1, completionCode = 0; !completionCode;)
   {
      completionCode = ScanBinderyObject (searchObjectName,
          searchObjectType, &objectID, objectName,
          &objectType, &objectHasProperties,
          &objectFlag, &objectSecurity);
      if (completionCode == 0)
      {
         printf ("%15s             %2d               %8lX\n",
                 objectName, objectType, objectID);
         printf ("More %d, Dynamic/Static %d, R/W Access %d\n",
                 objectHasProperties, objectFlag,
                 objectSecurity);
      }
   }
}
```

ScanBinderyObjectTrusteePaths

Returns the paths to which an object has trustee rights (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWScanObjectTrusteePaths (page 80))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwdir.h>

int ScanBinderyObjectTrusteePaths (
   LONG   objectID,
   BYTE   volumeNumber,
   int    *sequenceNumber,
   WORD   *trusteeAccessMask,
   char   *trusteePathName);
```

Parameters

objectID

(IN) Specifies a unique bindery object ID for which trustee information should be found.

volumeNumber

(IN) Specifies the volume number of the volume to be searched (0 to 31).

sequenceNumber

(IN) Contains the sequence number from the previous search (initial search requires a -1).

trusteeAccessMask

(OUT) Receives the object's trustee rights to `trusteePathName` (the returned path).

trusteePathName

(OUT) Receives a string containing a path of which the object is a trustee. The path is in the form:

```
volume:directory\...\directory | file
```

The maximum is 319 characters (MAX_SERVER + MAX_VOLUME + MAX_PATH). If the buffer allocated to this parameter is smaller than 319 characters, the server can abend if the buffer is overwritten.

Return Values

0	(0x00)	ESUCCESS
137	(0x89)	ERR_NO_SEARCH_PRIVILEGE
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
156	(0x9C)	ERR_NO_MORE_TRUSTEES
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
241	(0xF1)	ERR_INVALID_BINDERY_SECURITY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

This function is used iteratively to scan and return all of the paths (directories and files) and the corresponding access masks for which the specified object is a trustee.

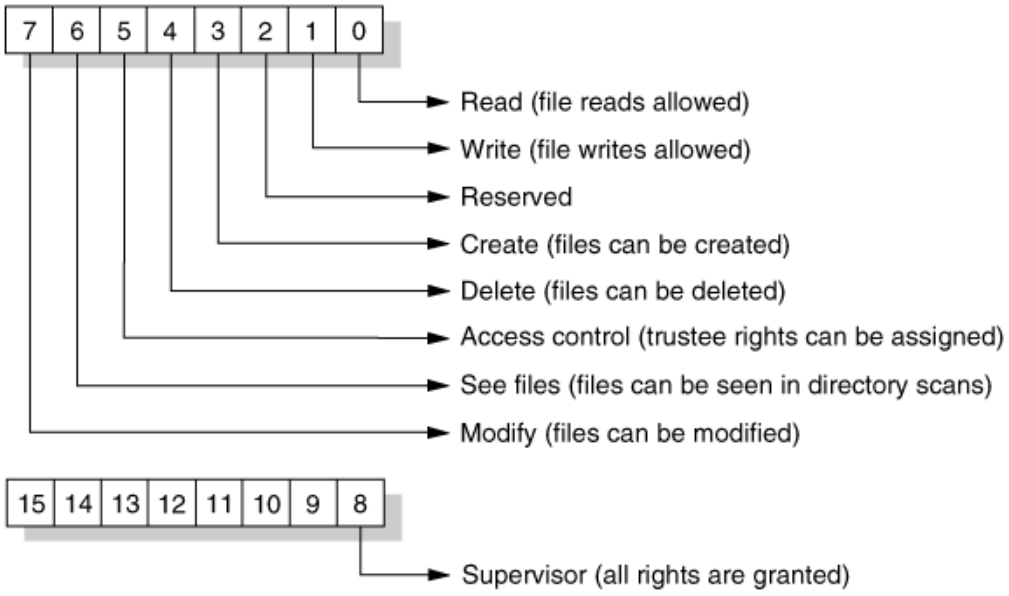
The `sequenceNumber` parameter should initially be set to -1 to get the first trustee path. Upon return, the sequence number is set to the value needed for the next call. Do not modify this value as this function is iteratively called to obtain all of the trustee's paths.

When all valid trustee paths have been returned, ERR_NO_MORE_TRUSTEES is returned and `trusteePathName` is set to "\0."

Only the supervisor, the object, or a bindery object that is security equivalent to the supervisor or object, can scan an object's trustee paths.

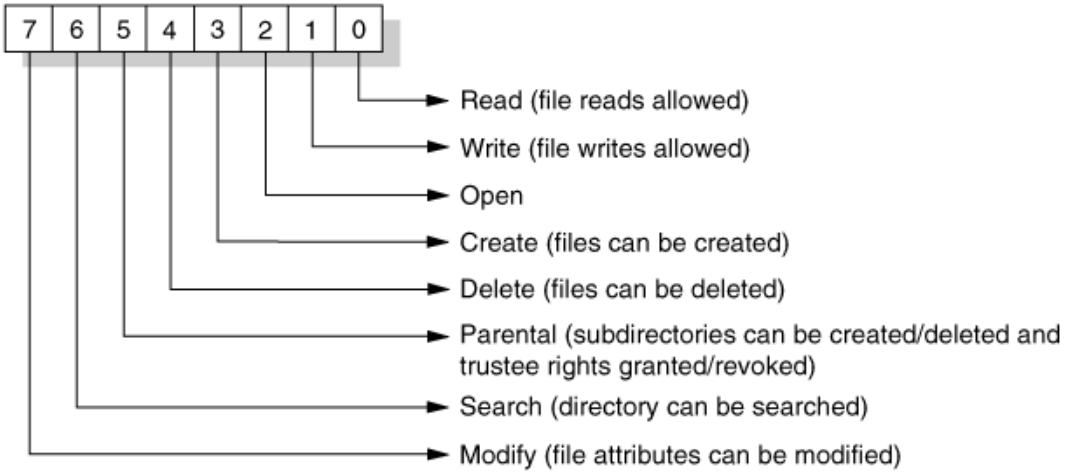
The `trusteeAccessMask` gives the trustee specific rights within the directory or the file, and if it is a directory, in all that directory's subdirectories, unless the trustee's rights are explicitly disallowed (using the Inherited Rights Mask) in those directories.

Figure 6-2 *The trusteeAccessMask for NetWare 3.0 and above*



For versions of NetWare previous to 3.0, the trustee rights appear in a 1-byte format as follows:

Figure 6-3 *The 1-byte trustee rights mask for NetWare 2.x*



See Also

GetVolumeNumber (NetWare SDK)

ScanBinderyObjectTrusteePaths Example

```
#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <nwconio.h>
#include <nwtypes.h>
#include <nwdir.h>

main()
{
   int rc,i;
   long oid;
   int sn;
   WORD tam;
   char path[1000];

   printf("object id to scan (hex): ");
   scanf("%x",&oid);
   i = 0;
   sn = 0;
   while(!(rc = ScanBinderyObjectTrusteePaths
        (oid,0,&sn,&tam,path)))
   {
      i++;
      printf("sequence number = %d\n",sn);
      printf("access mask = %#X\n",tam);
      printf("path name is\n%s\n\n",path);
   }
   printf("number of paths = %d\n",i);
   printf("rc = %x\n",rc);
}
```

ScanProperty

Scans the bindery for an object's properties (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWScanProperty (page 85))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int ScanProperty (
   char   *objectName,
   WORD   objectType,
   char   *searchPropertyName,
   long   *sequenceNumber,
   char    *propertyName,
   char   *propertyFlags,
   char   *propertySecurity,
   char   *propertyHasValue,
   char   *moreProperties);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object to search (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

searchPropertyName

(IN) Specifies the string containing the name of the property to search for (maximum 16 characters; can contain wildcard characters, including the NULL terminator).

sequenceNumber

(IN/OUT) Contains the sequence number from the previous search (initial search requires a -1) and receives the sequence number for the matching property.

propertyName

(OUT) Receives a string containing the name of the matching property (maximum 16 characters, including the NULL terminator).

propertyFlags

(OUT) Receives the property flags of the matching property: BF_DYNAMIC or BF_STATIC is logically ORed with BF_ITEM or BF_SET (BF_STATIC |BF_ITEM, and so on).

propertySecurity

(OUT) Receives the read and write security of the matching property.

propertyHasValue

(OUT) Receives a flag that indicates if the property has an attached value that can be read (0 = no value for property, 255 = property has value).

moreProperties

(OUT) Receives a flag that indicates if the bindery object has more properties (0 = no more properties for given object, 255 = more properties to scan).

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
241	(0xF1)	ERR_INVALID_BINDERY_SECURITY
251	(0xFB)	ERR_NO_SUCH_PROPERTY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

This function is used iteratively to scan the bindery for all properties of the bindery object that match the `searchPropertyName` parameter. This function passes the `objectName`, `objectType`, `searchPropertyName`, and `sequenceNumber` parameters. The function returns the `sequenceNumber`, `propertyName`, `propertyFlags`, `propertySecurity`, `propertyHasValue`, and `moreProperties` parameters.

The `objectName` and `objectType` parameters must uniquely identify the bindery object and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `sequenceNumber` parameter should be set to -1 for the first search. Upon return, the `moreProperties` flag is set if the matched property is not the last property. If it is not the last property, the `sequenceNumber` receives a number to be used as the `sequenceNumber` for the next call.

The `propertyFlags` parameter indicates whether the property is Dynamic or Static and whether it is of type Set or Item. A Dynamic property is a property that is created and deleted frequently. Dynamic properties are deleted from the bindery when the server is initialized. Static properties remain in the bindery until deleted with the DeleteProperty function.

The property type indicates the type of data stored in the Value of a property. The value of a Set property contains a series of bindery object IDs. Each object ID is 4 bytes long. The value of an Item property contains segments of 128-byte strings.

The `propertySecurity` parameter is actually two nibbles. The low-order nibble determines who can scan for and find the object. The high-order nibble determines who can add properties to the object.

The following values are defined for each nibble:

0	0 0 0 0	Anyone
1	0 0 0 1	Logged
2	0 0 1 0	Object
3	0 0 1 1	Supervisor
4	0 1 0 0	NetWare Operating System

For example, 0x31 indicates that any user logged in to the server can find the object, but only the supervisor can add a property to the object.

See Also

ReadPropertyValue (page 144), ScanBinderyObject (page 151)

ScanProperty Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
int   completionCode;
WORD  objectType;
long  sequenceNumber;
char  objectName[48], searchPropertyName[16], propertyName[16];
char  propertyFlags, propertySecurity, propertyHasValue;
char  moreProperties;

   printf ("\n\n");
   printf ("Enter Object Name -> ");
   scanf ("%s", objectName);
   printf ("\nEnter Object Type -> ");
   scanf ("%d", &objectType);
   printf ("\nEnter Property Name to Search -> ");
   scanf ("%s", searchPropertyName);
   sequenceNumber = -1;
   moreProperties = 0;
   do
   {
      completionCode = ScanProperty (objectName, objectType,
               searchPropertyName, &sequenceNumber,
```

```
                propertyName, &propertyFlags, &propertySecurity,
                &propertyHasValue, &moreProperties);
      printf ("\n\n\n");
      if (completionCode)
      {
         if (completionCode == 150)
            printf ("SERVER OUT OF MEMORY\n");
         else if (completionCode == 241)
            printf ("INVALID BINDERY SECURITY\n");
         else if (completionCode == 251)
            printf ("NO SUCH PROPERTY\n");
         else if (completionCode == 252)
            printf ("NO SUCH OBJECT\n");
         else if (completionCode == 254)
            printf ("SERVER BINDERY LOCKED\n");
         else if (completionCode == 255)
            printf ("BINDERY FAILURE\n");
         else
            printf ("completionCode = %d\n", completionCode);
      }
      else
      {
         printf (" SUCCESSFULLY COMPLETED...\n\n\n");
         printf ("Property Name...       %s\n",
                  propertyName);
         printf ("Property Flags...      %d\n",
                  propertyFlags);
         printf ("Property Security...   %2X\n",
                  propertySecurity);
         printf ("Property has Value..   %d\n",
                  propertyHasValue);
         printf ("More Property...       %d\n",
                  moreProperties);
      }
   } while (moreProperties != 0);
}
```

VerifyBinderyObjectPassword

Verifies that the password of a bindery object is valid (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWVerifyObjectPassword (page 88))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int VerifyBinderyObjectPassword (
   char   *objectName,
   WORD   objectType,
   char   *password);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

password

(IN) Specifies the string containing the password to be checked (maximum 128 characters; must be uppercase; NULL string = no password).

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
251	(0xFB)	ERR_NO_SUCH_PROPERTY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0XFF)	ERR_NO_SUCH_OBJECT_OR_BAD_PASSWORD

Remarks

This function verifies the password of a bindery object by passing the `objectName`, `objectType`, and `password` parameters.

The `objectName` and `objectType` parameters must uniquely identify the bindery object and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

There is a distinction between a bindery object without a password property and a bindery object with a password property that has no value. An entity is not allowed to log in to a server as a bindery object that does not have a password property. However, an entity is allowed to log in to a server as a bindery object that has a password property with no value. This function does not require that the requesting entity be logged in to the server.

If the object does not have a password property then the check fails. If the object has a password of length zero, a password of length zero matches.

See Also

ChangeBinderyObjectPassword (page 108)

WritePropertyValue

Writes a value to a property of type Item (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWWritePropertyValue (page 90))

Local Servers: blocking

Remote Servers: blocking

Classification: 3.x, 4.x, 5.x, 6.x

Service: Server-Based Bindery

Syntax

```
#include <\nlm\nit\nwbindry.h>

int WritePropertyValue (
   char   *objectName,
   WORD   objectType,
   char   *propertyName,
   int    segmentNumber,
   BYTE   *propertyValue,
   BYTE   moreSegments);
```

Parameters

objectName

(IN) Specifies the string containing the name of the bindery object (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the bindery object (OT_USER, OT_GROUP, OT_PRINT_SERVER, and so on).

propertyName

(IN) Specifies the string containing the name of the property to which the data is to be written (maximum 16 characters, including the NULL terminator).

segmentNumber

(IN) Segment number of the data to be written: 1 = First segment of the property's value.

propertyValue

(IN) Contains a segment of the property's value (maximum 128 bytes).

moreSegments

(IN) Indicates if the property value has more data segments after the current segment: 0 = No more segments 255 = More segments follow.

Return Values

0	(0x00)	ESUCCESS
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY
232	(0xE8)	ERR_NOT_ITEM_PROPERTY
236	(0xEC)	ERR_NO_SUCH_SEGMENT
240	(0xF0)	ERR_WILDCARD_NOT_ALLOWED
241	(0xF1)	ERR_INVALID_BINDERY_SECURITY
248	(0xF8)	ERR_NO_PROPERTY_WRITE_PRIVILEGE
251	(0xFB)	ERR_NO_SUCH_PROPERTY
252	(0xFC)	ERR_NO_SUCH_OBJECT
254	(0xFE)	ERR_SERVER_BINDERY_LOCKED
255	(0xFF)	ERR_BINDERY_FAILURE

Remarks

The `objectName`, `objectType`, and `propertyName` parameters must uniquely identify a bindery object's property and must not contain wildcard characters. The `objectName` can be from 1 to 48 characters long, including the NULL terminator. The `propertyName` can be from 1 to 16 characters long, including the NULL terminator. Only printable characters can be used. Slashes, backslashes, colons, semicolons, commas, asterisks, and question marks are prohibited.

The `segmentNumber` parameter should be set to 1 to write the first data segment. When creating a property value, the segments must be written in sequential order. Before segment `n` can be written, all segments from 1 to `n`-1 must have been written. To write property data of more than one segment (128 bytes), this function should be called iteratively. Once all segments of a property value have been established, segments can be written at random.

When writing the last segment of a property value the `moreSegments` flag should be set to zero. The bindery truncates the property value and discards extra segments if the `moreSegments` flag is 0 and the bindery has segments beyond the segment written. Property values should be kept to a single segment (128 bytes) to improve bindery efficiency.

The bindery makes no attempt to coordinate activities between multiple entities that might be reading or writing data to a single property. This means that one entity might read a partially-updated property and get inconsistent data, if the property's data extends across multiple segments. If this presents a problem, coordination on reads and writes must be handled by application programs. Logical record locks can be used to coordinate activities among applications.

Do not use this function to write values to set properties. Instead, call AddBinderyObjectToSet.

See Also

AddBinderyObjectToSet (page 104), ReadPropertyValue (page 144)

WritePropertyValue Example

```
#include <stdio.h>
#include <\nlm\nit\nwbindry.h>

main()
{
int   completionCode, segmentNumber;
char  objectName[48], propertyName[16];
WORD  objectType;
BYTE  propertyValue[128], moreSegments;

   printf ("\n\n");
   printf ("Enter Object Name -> ");
   scanf ("%s", objectName);
   printf ("\nEnter Object Type -> ");
   scanf ("%d", &objectType);
   printf ("\nEnter Property Name -> ");
   scanf ("%s", propertyName);
   printf ("\nEnter Segment Number -> ");
   scanf ("%d", &segmentNumber);
   printf ("\nEnter Property Value -> ");
   scanf ("%d", &propertyValue);
   printf ("\nEnter More Segments -> ");
   scanf ("%d", &moreSegments);
   completionCode = WritePropertyValue (objectName,
      objectType, propertyName, segmentNumber,
      propertyValue, moreSegments);
   printf ("\n\n\n");
   if (completionCode)
   {
      if (completionCode == 150)
         printf ("SERVER OUT OF MEMORY\n");
      else if (completionCode == 232)
         printf ("NO ITEM PROPERTY\n");
      else if (completionCode == 236)
         printf ("NO SUCH SEGMENT\n");
      else if (completionCode == 240)
         printf ("WILDCARD NOT ALLOWED\n");
      else if (completionCode == 241)
         printf ("INVALID BINDERY SECURITY\n");
      else if (completionCode == 248)
         printf ("NO PROPERTY WRITE PRIVILEGE\n");
      else if (completionCode == 251)
         printf ("NO SUCH PROPERTY\n");
      else if (completionCode == 252)
         printf ("NO SUCH OBJECT\n");
      else if (completionCode == 254)
         printf ("SERVER BINDERY LOCKED\n");
      else if (completionCode == 255)
         printf ("BINDERY FAILURE\n");
      else
         printf ("completionCode = %d\n", completionCode);
   }
```

```
   else
      printf (" SUCCESSFULLY COMPLETED...\n");
}
```

Revision History

A

The following table outlines all the changes that have been made to the Bindery documentation (in reverse chronological order):

Release Date	Revision Description
March 1, 2006	Updated format.
October 5, 2005	Transitioned to revised Novell documentation standards.
March 2, 2005	Updated legal information.
June 9, 2004	Added documentation for two new UTF-8 functions: NWGetObjectEffectiveRightsExt (page 63) and NWScanObjectTrusteePathsExt (page 83)
February 2002	Updated Pascal syntaxes.
	Updated links.
September 2001	Added NetWare 6.x support to documentation.
	Added alternative text to figures.
June 2001	Made changes to improve document accessibility.
	Added links to Section 5.4, “Server-Based Bindery Functions,” on page 101.
May 2000	Added this revision history