Data General
Data General Corporation, Westboro, Massachusetts 01580

Customer Documentation

NetWare® for AViiON® Systems: C Interface Reference Guide

069–000567–00

AViiON®
PRODUCT LINE

NetWare® for AViiON® Systems: C Interface Reference Guide

069–00567–00

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085–series) supplied with the software.

Rev. 00, January 1992
Licensed Material — Property of the copyright holder(s)
Ordering No. 069–000567

Notice

NetWare® for AViiON® Systems:
C Interface Reference Guide
069–000567–00

Revision History:

Original Release – January 1992

Effective with:

NetWare® for AViiON® Systems,
Revision 1.30

Preface

This manual provides a systematic and comprehensive reference to the Applications Programming Interface (API) library of functions and calls for NetWare® for AViiON® Systems. For more general background information about API functions and services, please see the companion manual, *NetWare® for AViiON® Systems: C Interface Programmer's Guide* (069–000566).

All references to software versions in this manual are inclusive; no distinction is made between different releases of the same version. In other words, we refer to all NetWare 286 products as NetWare 2.x, and we refer to all NetWare 386 products as NetWare 3.x. NetWare for AViiON Systems is compatible with NetWare 3.x unless otherwise noted. The release notice accompanying your shipment provides the most current information about exceptions to this compatibility.

IMPORTANT: Unless otherwise noted, NetWare for AViiON Systems servers cannot service API programs running in a NetWare network.

Organization of this manual

Chapter 1 Accounting Service APIs

This chapter explains how to use the Accounting Service APIs that allow file servers to charge clients for services.

Chapter 2 Bindery Service APIs

This chapter explains how to use the Bindery Service APIs that regulate access to the file server.

Chapter 3 Connection Service APIs

This chapter explains how to use Connection Service APIs to establish and destroy logical connections to the file server and control the return of status information about those connections.

Chapter 4 File Service APIs

This chapter explains how to use File Service APIs to enable applications to manipulate files, directories, volumes, trustees, and their associated information.

Chapter 5 Path Service APIs

This chapter explains how to use Path Service APIs to allocate directory handles and return information about directory paths.

Chapter 6 Queue Management Service APIs

This chapter explains how to use Queue Management Service APIs to control the flow of jobs and services on the network.

Chapter 7 Server Platform Service APIs

This chapter explains how to use Server Platform Service APIs in function calls that report file server information and disk usage.

Chapter 8 Synchronization Service APIs

This chapter explains how to use Synchronization Service APIs to coordinate access to network files and other resources.

Chapter 9 Transaction Tracking Service APIs

This chapter explains how to use Transaction Tracking Service APIs to ensure file integrity of selected files.

Appendix A Constant Declarations and Structure Definitions

This appendix defines constants and provides structure requirements for all API calls.

Appendix B NetWare Errors

This appendix lists all NetWare error codes and explains their interpretation.

Appendix C DG/UX Errors

This appendix explains how to avoid failures of C calls caused by the two DG/UX™ interrupt signals, SIGPOLL and SIGALRM.

Appendix D Differences

This appendix describes the differences between the NetWare for AViiON Systems API library and the API library previously released with NetWare C Interface-DOS.

Related Documents

You received a comprehensive set of documents with your NetWare for AViiON Systems release package. The manuals listed below are included in that set and contain information that augments the text of this manual.

NetWare® for AViiON® Systems: Concepts (069–000483)

This manual provides an alphabetically-arranged glossary of NetWare terminology. It is written for all levels of NetWare users, but it will be particularly useful to supervisors who are performing their first installation of the NetWare for AViiON Systems product.

NetWare® for AViiON® Systems: C Interface Programmer's Guide (069–000566)

This manual provides a background of information for NetWare applications programmers and a general overview of available API services and functions. It is written specifically for applications programmers.

NetWare® for AViiON® Systems: System Administration (069–000487)

This manual provides a reference to the SCONSOLE and HYBRID utilities and the NetWare for AViiON Systems printing services and utilities. It is written primarily for network supervisors who will use SCONSOLE and HYBRID to administer the AViiON file server and set up DG/UX printers using the NetWare printing utilities.

NetWare®for AViiON® Systems: Installation (069–000488)

This manual provides detailed instructions for planning a NetWare network, installing NetWare for AViiON Systems on an AViiON computer, configuring client workstations, and setting up user accounts. It is written for the network supervisor.

NetWare® for AViiON® Systems: User Book (069–000486)

This manual provides a general overview of NetWare. It is written for first-time users who are unfamiliar with networks.

NetWare® for AViiON® Systems: Utilities (069–000484)

This manual provides an alphabetically-arranged reference for NetWare command line and menu utilities. It is written for all levels of NetWare users.

Reader, please note

In all examples within the text, we use

```
This typeface to show system prompts and responses.
```

To show which NetWare products support a given call, we use a chart similar to the following:

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

Contacting Data General

Data General wants to assist you in any way it can to help you use its products. Please feel free to contact the company as outlined below.

Manuals

If you require additional manuals, please contact your local Data General sales representative.

Telephone assistance

If you are unable to solve a problem with your system, free telephone assistance is available with your warranty and with most Data General service options. If you are within the United States or Canada, contact the Data General Customer Support Center (CSC) by calling 1–800–DG–HELPS. Lines are open from 8:00 a.m. to 5:00 p.m., your time, Monday through Friday. The center will put you in touch with a member of Data General's telephone assistance staff who can answer your questions.

For telephone assistance outside the United States or Canada, ask your Data General sales representative for the appropriate telephone number.

Joining our user's group

Please consider joining the largest independent organization of Data General users, the North American Data General Users Group (NADGUG). In addition to making valuable contacts, members receive *FOCUS* monthly magazine, a conference discount, access to the Software Library and Electronic Bulletin Board, an annual *Member Directory*, Regional and Special Interest Groups, and much more. For more information about membership in the North American Data General Users Group, call 1–800–253–3902 or 1–508–443–3330.

End of Preface

Contents

Chapter 4 File Service APIs

Chapter 5 Path Service APIs

Chapter 6 Queue Management Service APIs

Chapter 7 Server Platform Service APIs

Chapter 8 Synchronization Service APIs

Chapter 1
Accounting Service APIs

Function Calls

This chapter describes the following Accounting Service APIs.

Introduction to Accounting Services

The four accounting service calls enable developers to create DG/UX™ servers that can charge for their services. For example, a database server can charge for the number of records viewed, the number of requests serviced, or the amount of time connected. A print server can charge for the number of pages printed.

For a server to charge for services, the server must be a member of the ACCOUNT_SERVERS property of the file server. See "Accounting Services" in *NetWare® for AViiON® Series Systems C Interface Programmer's Guide.*

To use accounting services, you must be familiar with the NetWare® file server bindery. See "Introduction to Bindery Services" in Chapter 2 for an explanation of Bindery objects, properties, and values.

NWGetAccountStatus

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the account status of a bindery object.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
uint16          objectType;
char            objectName[NWMAX_OBJECT_NAME_LENGTH];
int32           balance;
int32           limit;
NWHoldInfo_t    holds[NWMAX_NUMBER_OF_HOLDS];

ccode=NWGetAccountStatus( serverConnID, objectType, objectName,
    &balance, &limit, holds );
```

Input

serverConnID	Passes the file server connection ID.
objectType	Passes the type of bindery object for which the request is being made. (See Appendix A, Bindery Object Types.)
objectName	Passes a pointer to the string containing the object name for which the account status request is being made.
balance	Passes a pointer to the space allocated for the number of value units available to the object to buy services on the network.
limit	Passes a pointer to the space allocated for the value of the lowest level the object's account balance can reach before the object can no longer buy services on the network.
holds	Passes a pointer to the structure allocated for the list of objectIDs and holdAmounts that have been placed on the account (maximum = 16). (See Appendix A, NWHoldInfo_t Structure.)

Output

balance	Receives the number of value units available to the object to buy services on the network.
limit	Receives the value of the lowest level the object's account balance can reach before the object can no longer buy services on the network.
holds	Returns a list of objectIDs and holdAmounts that have been placed on the account (maximum = 16). (See Appendix A, NWHoldInfo_t Structure.)

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC0	No Account Privileges
0xC1	No Account Balance
0xC4	Account Disabled
0xEA	No Such Member

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function queries a file server's bindery for the current account status of a specified bindery object by passing the bindery object name and type. The function returns the object's balance, limit, and holds parameters.

The value in the balance parameter represents the object's account balance, usually in some established monetary unit such as cents.

The holds parameter lists servers that have issued NWSubmitAccountHold calls against the object and the amount reserved by each value-added server. The holds parameter is also lists the object ID number of a value-added server that has issued a NWSubmitAccountHold call against the object. Up to 16 servers can place holds on the account at one time. Multiple holds from the same server are combined. Each server hold is made up of two fields: (1) the object ID of the server that placed the hold, and (2) the amount of that server's hold.

See Also

NWSubmitAccountHold

NWSubmitAccountCharge

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function updates the account of a bindery object by charging for a service and updating the audit record.

Synopsis

```
#include "nwapi.h"

int         code;
uint16      serverConnID;
uint16      objectType;
char        objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16      serviceType;
int32       chargeAmount;
int32       cancelHoldAmount;
uint16      commentType;
char        comment[NWMAX_COMMENT_LENGTH];

ccode=NWSubmitAccountCharge( serverConnID, objectType, objectName,
      serviceType, chargeAmount, cancelHoldAmount, commentType, comment );
```

Input

serverConnID	Passes the file server connection ID.
objectType	Passes the type of bindery object for which the request is being made. (See Appendix A, Bindery Object Types.)
objectName	Passes a pointer to the name of the object for which the account status request is being made
serviceType	Passes the type of service for which the request is being made (usually the object type of the charging account server).
chargeAmount	Passes the amount of account server charges for the service it provides.
cancelHoldAmount	Passes the amount to be subtracted from the total amount of all holds previously placed by the server. If no NWSubmitAccountHold calls were made prior to providing the service, this value should be zero.
commentType	Passes the type of comment written to the audit report.
comment	Passes a pointer to a comment associated with the object's account charge.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x94	No Write Privileges
0xA2	I/O Lock Error

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function charges an object's account balance and relinquishes a hold against the object's account balance. The function can also write a note about the transaction in an audit record (optional). The charge and hold amounts do not have to be the same.

The objectType and objectName parameters must uniquely specify the bindery object and cannot contain wildcard characters.

The serviceType parameter usually contains the object type of the charging account server. The common server object types are listed below:

Object	Type
Archive Server	NWOT_ARCHIVE_SERVER
Job Server	NWOT_JOB_SERVER
Print Server	NWOT_PRINT_SERVER

See "Introduction to Bindery Services" in Chapter 2 for additional object types.

The commentType parameter contains the number of the comment type in the comment parameter. Comment types are administered by Data General and are listed below:

Comment Type	Description
1	Connect time charge
2	Disk storage charge
3	Log in note
4	Log out note
5	Account locked note
6	Server time modified note

Developer's should contact their Data General service representative for unique comment types. Comment types greater than 8000h are reserved for experimental purposes.

Notes

The comment parameter is the entry that the value-added server makes in an audit record. This audit record is contained in the SYS:SYSTEM\NET$ACCT.DAT file.

See Also

NWSubmitAccountNote

NWSubmitAccountHold

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function reserves a specified amount of an object's account balance pending a NWSubmitAccountCharge call.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint16      objectType;
char        objectName[NWMAX_OBJECT_NAME_LENGTH];
int32       reserveAmount;

ccode=NWSubmitAccountHold( serverConnID, objectType, objectName,
    reserveAmount );
```

Input

serverConnID	Passes the file server connection ID.
objectType	Passes the type of bindery object for which the request is being made. (See Appendix A, Bindery Object Types.)
objectName	Passes a pointer to the name of the bindery object for which the account status request is being made.
reserveAmount	Passes the hold amount to be placed against the client's account pending service.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x94	No Write Privileges
0xA2	I/O Lock Error
0xC1	No Account Balance
0xC3	Too Many Holds

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function reserves a specified amount of an object's account balance before that object receives and is charged for a service on the network.

The objectType and objectName parameters must uniquely identify the bindery object and may not contain wildcard characters.

The reserveAmount parameter gets the amount that the server expects to charge for the service it is about to provide to the object.

Notes

No more than 16 servers can reserve amounts of an object's account balance at one time. Multiple holds from the same server are combined.

NWSubmitAccountNote

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function adds a note about an object's account to an audit record. This API does not charge for the service.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint16    objectType;
char      objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16    serviceType;
uint16    commentType;
char      comment[NWMAX_COMMENT_LENGTH];

ccode=NWSubmitAccountNote( serverConnID, objectType, objectName,
    serviceType, commentType, comment );
```

Input

serverConnID	Passes the file server connection ID.
objectType	Passes the type of bindery object for which the request is being made. (See Appendix A, Bindery Object Types.)
objectName	Passes a pointer to the object name for which the account status request is being made.
serviceType	Passes the type of service for which the request is being made (usually the object type of the charging account server).
commentType	Passes the type of comment in the comment parameter. (See Appendix A, Comment Types.)
comment	Passes a pointer to the comment associated with the object's account.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xEA	No Such Member
0xEB	Not Set Property
0xEC	No Such Set
0xFC	No Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function adds a note about an accounting transaction to an audit record.

The objectType and objectName parameters must uniquely identify the bindery object and may not contain wildcard characters.

The serviceType parameter usually contains the object type of the charging account server. The common server object types are listed below:

Object	Type
Archive Server	NWOT_ARCHIVE_SERVER
Job Server	NWOT_JOB_SERVER
Print Server	NWOT_PRINT_SERVER

The commentType parameter contains the number of the comment type in the comment parameter. Comment types are administered by Data General and are listed below:

Comment Type	Description
1	Connect time charge
2	Disk storage charge
3	Log in note
4	Log out note
5	Account locked note
6	Server time modified note

Developer's should contact their Data General service representative for unique comment types. Comment types greater than 8000h are reserved for experimental purposes.

Notes

The comment parameter contains the entry that the server makes in the audit record. The audit record is contained in the SYS:SYSTEM\NET$ACCT.DAT file.

End of Chapter

Chapter 2
Bindery Service APIs

Function Calls

This chapter contains a description of the following Bindery Service APIs.

Introduction to Bindery Services

Each NetWare file server includes a small database or bindery implemented as hidden files. NetWare for AViiON Systems has three hidden bindery files (NET$OBJ.SYS, NET$VAL.SYS, and NET$PROP.SYS) which are located in SYS:SYSTEM. Within the Bindery, the NetWare operating system maintains a list of all objects (entities) allowed to access the file server. NetWare also records information about each bindery object.

Bindery Objects

A bindery object can be a user, user group, file server, print server, or any other named entity that can access a file server. Each bindery object consists of the following components.

Object Name — A 48-byte, null-terminated string that contains the name of the object. Only printable characters can be used. An object name cannot include spaces or the following characters:

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

Object ID — A 4-byte number that uniquely identifies the object within a particular file server's bindery. The NetWare operating system, not the application, assigns this number.

Object State — A 1-byte flag that specifies whether the object is Static (0x00) or Dynamic (0x01). A Static object exists in a bindery until an application intentionally deletes it with the NWDeleteObject function. A Dynamic object disappears from a file server's bindery when the file server is rebooted. (In the case of an object that is a service-advertising server, the object disappears from a bindery when the server ceases to advertise.)

Object Type — A 2-byte number that classifies an object as a user, user group, file server, etc. The following is a list of common object types:

Table 2-1 Object Types

Description	Object Type
Unknown	0x0000
User	0x0001
User Group	0x0002
Print Queue	0x0003
File Server	0x0004
Job Server	0x0005
Gateway	0x0006
Print Server	0x0007
Archive Queue	0x0008
Archive Server	0x0009
Job Queue	0x000A
Administration	0x000B
SNA Gateway	0x0021
Remote Bridge Server	0x0024
Synchronization Server	0x002D
Archive Server (Dynamic SAP)	0x002E
Advertizing Print Server	0x0047
Btrieve VAP	0x0050
Print Queue User	0x0053
NVT Server	0x009E
Wild	0xFFFF

Properties Flag

A 1-byte flag that indicates whether one or more properties are associated with the object.

0x00 = no associated properties
0xFF = one or more associated properties

Object Security

A 1-byte flag that determines access to 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. Refer Table 2-2 for the values defined for each nibble.

Table 2-2 Security Levels

Hex	Binary	Access	Description
0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the file server.
1	0001	Logged	Access allowed only to clients who have logged in to the file server.
2	0010	Object	Access allowed only to clients who have logged in to the file server with the object's name, type, and password.
3	0011	Supervisor	Access allowed only to clients who have logged in to the server as the supervisor or as an object that has supervisor security equivalence.
4	0100	NetWare	Access only allowed to the NetWare operating system.

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

NOTE:

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

Properties and Values

Each bindery object can have one or more properties associated with it. For example, the object DAN (object type 0x0001, user) might be associated with the properties GROUPS_I'M_IN, ACCOUNT_BALANCE, and PASSWORD. Note that GROUPS_I'M_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, the value of the GROUPS_I'M_IN property would be the object ID of a user group to which DAN belongs. The value of the property ACCOUNT_BALANCE would be user DAN's current balance. The value of the PASSWORD property would be DAN's login password.

Properties fall into one of the following two categories: item or set. These categories are described below.

Item Property. An Item property is made up of a 128-byte value. For example, the property ACCOUNT_BALANCE is an Item property that contains a monetary balance in the first few bytes of a 128-byte string and zeros in the rest.

Set Property. A Set property contains 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_I'M_IN is a Set property. The 128-byte segment associated with GROUPS_I'M_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.

A property consists of the following components: property name, property state, property type, property security, and values flag. These items are described below.

Property Name

A 15-byte string that contains the name of the property. A property name can contain only printable characters except any of the following:

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

Property State

A 1-byte field with bits 0 and 1 defined. Bit 0 is the Static/Dynamic flag defined as follows:

3 2 1 0	Bit number
0 0 0 0	Static
0 0 0 1	Dynamic

A Static property exists until it is explicitly deleted. A Dynamic property is deleted from the file server's bindery when the file server is rebooted.

Property Type

A 1-byte field with bits 0 and 1 defined. Bit 1 is the Item/Set flag defined as follows:

3 2 1 0	Bit number
0 0 0 0	Item
0 0 1 0	Set

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

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

0 0 1 0

Property Security

A 1-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 value(s) to the property (write security). 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

For example, 0x31 (0011 0001) indicates that any user logged in to the file server can find (read) the property, but only SUPERVISOR can add (write) values to the property.

Values Flag

A 1-byte flag that indicates whether an item property has more than one value associated with it. The following values are defined for the byte:

0000 0000	One value
1111 1111	More values

Using Property Values

The following charts list the APIs that need to be used to create properties, verify written values, delete property values, and delete properties.

Chart 1: Create Properties

Step	Type	API
Create the object (if the object does not exist)	Set Item	NWCreateObject (specifies object type)
Create the property (if the property does not exist)	Set Item	NWCreateProperty (specifies the object type that can use the property)
Write value to property	Set	NWAddObjectToSet
	Item	NWWritePropertyValue

Chart 2: Verify Written Values

Step	Type	API
Read value of property	Set	NWIsObjectInSet
	Item	NWScanPropertyValue

Chart 3: Delete Property Values

Step	Type	API
Delete a property value	Set	NWDeleteObjectFromSet
	Item	NWWritePropertyValue (overwrites existing value)

Chart 4: Delete Properties

Step	Type	API
Delete a property	Set Item	NWDeleteProperty

NWAddObjectToSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function adds a member to a bindery property of type SET.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      objectName[NWMAX_OBJECT_NAME_LENGTH);
uint16    objectType;
char      propertyName[NWMAX_PROPERTY_NAME_LENGTH];
char      memberName[NWMAX_MEMBER_NAME_LENGTH];
uint16    memberType;

ccode=NWAddObjectToSet( serverConnID, objectName, objectType,
    propertyName, memberName, memberType );
```

Input

serverConnID	Passes the current session's file server connection ID.
objectName	Passes a pointer to the set's object name.
objectType	Passes the set's bindery object type. (See Appendix A, Bindery Object Types.)
propertyName	Passes a pointer to the set's property name.
memberName	Passes a pointer to the name of the previously-created bindery object being added to the set.
memberType	Passes the bindery type of the member being added. (See Appendix A, Bindery Object Types.)

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xE9	Member Exists
0xEA	No Such Member
0xF8	No Property Write
0xFC	No Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The objectName, objectType, and propertyName parameters must uniquely identify the property and cannot contain wildcard characters.

The memberName and memberType parameters must uniquely identify the bindery object to be added and cannot contain wildcard characters. This object must already exist within the bindery.

The property must be of type SET.

This function searches consecutive segments of the property's value for an open slot where it can record the unique bindery object identification of the new member. The new member is inserted into the first available slot. If no open slot is found, a new segment is created and the new member's unique bindery object identification is written into the first slot of the new segment. The rest of the segment is filled with zeros.

Notes

A client must have write access to the property to make this call.

For properties of type ITEM, the application must use NWWritePropertyValue.

See Also

NWIsObjectInSet
NWDeleteObjectFromSet

NWChangeObjectPassword

NetWare 2.x	NetWare 3.x	NetWare for AVIION Systems
✓	✓	✓

This function changes the password of a bindery object.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16    objectType;
char      oldPassword[NWMAX_PASSWORD_LENGTH];
char      newPassword[NWMAX_PASSWORD_LENGTH];

ccode=NWChangeObjectPassword( serverConnID, objectName, objectType,
    oldPassword, newPassword )
```

Input

serverConnID	Passes the file server connection ID.
objectName	Passes a pointer to the object name.
objectType	Passes the object type. (See Appendix A, Bindery Object Types.)
oldPassword	Passes a pointer to the old password.
newPassword	Passes a pointer to the new password.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xD7	Duplicate Password
0xF1	Bindery Security
0xF8	No Property Write
0xFC	No Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function creates or changes an object password. It also assigns the property security (0x44) to the property PASSWORD. The property security allows only the NetWare operating system to find, 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.

Notes

This is the only function call which can create or change object passwords. Although PASSWORD is a property, it is a unique property which can not be created with the NWCreateProperty function call.

There is a distinction between a bindery object without a password property and a bindery object with a password property that has no value. A workstation is not allowed to log in to a file server as a bindery object that does not have a PASSWORD property. However, a workstation is allowed to log in to a file server as a bindery object with a password with no value.

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

See Also

NWIsObjectPasswordOK

NWChangeObjectSecurity

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function changes the security access mask of a bindery object on the file server connected via the file server connection ID (serverConnID parameter).

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16    objectType;
uint8     newObjectSecurity;

ccode=NWChangeObjectSecurity( serverConnID, objectName, objectType,
    newObjectSecurity );
```

serverConnID — Passes the file server connection ID.

objectName — Passes a pointer to a string containing the object name.

objectType — Passes the type of the bindery object. (See Appendix A, Bindery Object Types.)

newObjectSecurity — Passes the new security access level for the specified object.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xF1	Invalid Bindery Security
0xF5	No Object Create
0xFC	No Such Object
0xFE	Bindery Locked

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The objectName and objectType parameters must uniquely identify the bindery object and cannot contain wildcard specifiers.

The newObjectSecurity parameter is a byte in which the low 4 bits (nibble) control read security and the high four bits control write security. Read security determines which clients can find the bindery object when they scan for it. Write security determines which clients can create properties for the bindery object. Table 2-2, above, describes this security level.

For example, a bindery object with a newObjectSecurity of 0x31 can be viewed by any client that has successfully logged in to the file server, but only a client with security equivalence to SUPERVISOR can add properties to it.

Read Security:

0xn0 = NWBS_ANY_READ
0xn1 = NWBS_LOGGED_READ
0xn2 = NWBS_OBJECT_READ
0xn3 = NWBS_SUPER_READ
0xn4 = NWBS_BINDERY_READ

Write Security:

0x0n = NWBS_ANY_WRITE
0x1n = NWBS_LOGGED_WRITE
0x2n = NWBS_OBJECT_WRITE
0x3n = NWBS_SUPER_WRITE
0x4n = NWBS_BINDERY_WRITE

This function cannot set or clear BINDERY read or write security.

Notes

Only SUPERVISOR or a bindery object that is security equivalent to SUPERVISOR can change a bindery object's security.

See Also

NWCreateObject

NWChangePropertySecurity

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function changes the security access mask of a property in a bindery object on the file server associated with the file server connection ID (serverConnID parameter).

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16    objectType;
char      propertyName[NWMAX_PROPERTY_NAME_LENGTH];
uint8     newPropertySecurity;

ccode=NWChangePropertySecurity( serverConnID, objectName, objectType,
    propertyName, newPropertySecurity );
```

Input

serverConnID	Passes the file server connection ID.
objectName	Passes a pointer to the name of the bindery object associated with the property whose security is being changed.
objectType	Passes the type of the object described by the objectName parameter. (See Appendix A, Bindery Object Types.)
propertyName	Passes a pointer to the name of the affected property.
newPropertySecurity	Passes the new security access level for the property.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xF2	No Object Read Privilege
0xF6	No Property Delete Privilege
0xFB	No Such Property
0xFC	No Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The objectName, objectType, and propertyName parameters must uniquely identify the property and cannot contain wildcard specifiers.

The newPropertySecurity is a byte in which the low 4 bits (nibble) control read security and the high 4 bits control write security. Read security determines which clients can read the property. Write security determines which clients can write to the property. See Table 2-2, above, for a description of each security level.

For example, a property with a newPropertySecurity of 0x31 can be seen by any client that has successfully logged in to the file server, but only a client with security equivalence to SUPERVISOR can write to the property.

Read Security:

0xn0 = NWBS_ANY_READ
0xn1 = NWBS_LOGGED_READ
0xn2 = NWBS_OBJECT_READ
0xn3 = NWBS_SUPER_READ
0xn4 = NWBS_BINDERY_READ

Write Security:

0x0n = NWBS_ANY_WRITE
0x1n = NWBS_LOGGED_WRITE
0x2n = NWBS_OBJECT_WRITE
0x3n = NWBS_SUPER_WRITE
0x4n = NWBS_BINDERY_WRITE

Notes

This function cannot set or clear BINDERY read or write security.

The requesting process cannot change a property's security to a level greater than the process's access to the property.

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

See Also

NWCreateObject
NWCreateProperty

NWCloseBindery

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function closes the bindery on the file server associated with the file server connection ID (serverConnID parameter).

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;

ccode=NWCloseBindery( serverConnID );
```

Input

serverConnID Passes the file server connection ID.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFF	Close Failure
0x96	Server Out Of Memory

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

Because the bindery files contain all information about the file server's clients, the bindery should be archived on a regular basis. However, the file server keeps bindery files open and locked at all times so that they cannot be accessed directly. For bindery files to be archived, the bindery must be closed with the NWCloseBindery function.

This function allows SUPERVISOR, or an object that has security equivalence to SUPERVISOR, to close and unlock the bindery files, thus allowing the bindery to be archived. After the bindery files have been archived, the NWOpenBindery function is used to give control of the bindery files back to the file server. While the bindery is closed, much of the functionality of the network is disabled.

See Also

NWOpenBindery

NWCreateObject

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function adds a new object to the bindery on the file server associated with the file server connection ID (serverConnID parameter).

The bindery object must have a password property to log in to a file server. The password property is created with the NWChangeObjectPassword function.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
char        newObjectName[NWMAX_OBJECT_NAME_LENGTH];
uint16      newObjectType;
uint8       newObjectState;
uint8       newObjectSecurity;

ccode=NWCreateObject( serverConnID, newObjectName, newObjectType,
    newObjectState, newObjectSecurity );
```

Input

serverConnID	Passes the server connection ID for the file server whose bindery is being affected.
newObjectName	Passes a pointer to the string containing the new object name.
newObjectType	Passes the bindery type of the new object. (See Appendix A, Bindery Object Types.)
newObjectState	Passes a flag indicating the object state. (See Appendix A, Bindery Object and Property States.)
newObjectSecurity	Passes the new object's access rights mask.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xEE	Object Exists
0xEF	Illegal Name
0xF1	Invalid Bindery Security
0xF5	No Object Create Privilege

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The newObjectName and newObjectType parameters must uniquely identify the bindery object and cannot contain wildcard specifiers.

Only SUPERVISOR or a bindery object that is security equivalent to SUPERVISOR can create bindery objects.

The newObjectSecurity is a byte in which the low 4 bits (nibble) control read security while the high 4 bits control write security. Read security determines which clients can find the bindery object when they scan for it. Write security determines which clients can create properties for the bindery object. The read and write values are described in Table 2-2, above.For example, a bindery object with a newObjectSecurity of 0x31 can be seen by any client that has successfully logged in to the file server, but only a client with security equivalence to SUPERVISOR can add properties to it.

Read Security:

0xn0 = NWBS_ANY_READ
0xn1 = NWBS_LOGGED_READ
0xn2 = NWBS_OBJECT_READ
0xn3 = NWBS_SUPER_READ
0xn4 = NWBS_BINDERY_READ

Write Security:

0x0n = NWBS_ANY_WRITE
0x1n = NWBS_LOGGED_WRITE
0x2n = NWBS_OBJECT_WRITE
0x3n = NWBS_SUPER_WRITE
0x4n = NWBS_BINDERY_WRITE

See Also

NWChangeObjectPassword
NWCreateProperty

NWCreateProperty

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function adds a property to a bindery object.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
char        objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16      objectType;
char        newPropertyName[NWMAX_PROPERTY_NAME_LENGTH];
uint8       newPropertyTypeAndState;
uint8       newPropertySecurity;

ccode=NWCreateProperty( serverConnID, objectName, objectType,
      newPropertyName, newPropertyTypeAndState, newPropertySecurity );
```

Input

serverConnID	Passes the server connection ID.
objectName	Passes a pointer to the object name receiving the new property.
objectType	Passes the type of the affected bindery object. (See Appendix A, Bindery Object Types.)
newPropertyName	Passes a pointer to the name of the property being created.
newPropertyTypeAndState	Passes the OR'ed value of the property type and the property state. (See Appendix A, Bindery Property Types and Bindery Object and Property States.)
newPropertySecurity	Passes the new property's security access mask.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xEE	Object Exists
0xEF	Illegal Name
0xF1	Bindery Security
0xF5	No Object Create

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The newPropertyTypeAndState parameter defines a property's type and state (dynamic or static). A dynamic property is one that is created and deleted frequently. Dynamic properties are deleted from the bindery when the file server is rebooted.

The property type indicates the type of data a property value contains. SET property types contain a set of bindery object identifications. The bindery attaches no significance to the contents of a property value if the property is of type ITEM. (See "Introduction to Bindery Services" in this chapter.)

The newPropertySecurity parameter is a byte in which the low 4 bits (nibble) control read security and the high 4 bits control write security. Read security controls which clients can read the property. Write security controls which clients can write to the property. The values for the newPropertySecurity parameter are described in Table 2-2, above.

For example, a property with the newPropertySecurity parameter equal to 0x31 can be seen by any client that has successfully logged in to the file server, but only a client with security equivalent to SUPERVISOR can write to the property.

Read Security:

0xn0 = NWBS_ANY_READ
0xn1 = NWBS_LOGGED_READ
0xn2 = NWBS_OBJECT_READ
0xn3 = NWBS_SUPER_READ
0xn4 = NWBS_BINDERY_READ

Write Security:

0x0n = NWBS_ANY_WRITE
0x1n = NWBS_LOGGED_WRITE
0x2n = NWBS_OBJECT_WRITE
0x3n = NWBS_SUPER_WRITE
0x4n = NWBS_BINDERY_WRITE

The requesting process cannot create properties that have security greater than the process's access to the bindery object.

The password property is created by calling NWChangeObjectPassword rather than by using the NWCreateProperty function.

Notes

The PASSWORD property can not be created with this function call. You must use NWChangeObjectPassword to create or change an object's password.

This function requires write access to the bindery object.

See Also

NWChangeObjectPassword
NWCreateObject

NWDeleteObject

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function removes an object from the bindery of the file server associated with the file server connection ID (serverConnID).

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16    objectType;

ccode=NWDeleteObject( serverConnID, objectName, objectType );
```

Input

serverConnID	Passes the file server connection ID.
objectName	Passes a pointer to the object name being deleted.
objectType	Passes the bindery type of the object being deleted. (See Appendix A, Bindery Object Types.)

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xF2	No Object Read
0xF4	No Object Delete
0xF6	No Property Delete
0xFC	No Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Notes

The objectName and objectType parameters must uniquely identify the bindery object and cannot contain wildcard specifiers. Only SUPERVISOR or a bindery object that is security equivalent to SUPERVISOR can delete bindery objects.

See Also

NWDeleteObjectFromSet

NWDeleteObjectFromSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function deletes a member from a bindery property of type SET on the file server associated with the file server connection ID (serverConnID).

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16    objectType;
char      propertyName[NWMAX_PROPERTY_NAME_LENGTH];
char      memberName[NWMAX_MEMBER_NAME_LENGTH];
uint16    memberType;

ccode=NWDeleteObjectFromSet( serverConnID, objectName, objectType,
    propertyName, memberName, memberType );
```

Input

serverConnID	Passes the file server connection ID.
objectName	Passes a pointer to the name of the bindery object whose set is being affected.
objectType	Passes the object type of bindery object whose set is being affected. (See Appendix A, Bindery Object Types.)
propertyName	Passes a pointer to the name of the property (of type SET) from which the member is being deleted.
memberName	Passes A pointer to the name of the bindery object that is being deleted from the set.
memberType	Passes the object type of the member being deleted. (See Appendix A, Bindery Object Types.)

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xEB	Property Not Set Property
0xF8	No Property Write
0xFB	No Such Property
0xFC	No Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

There are two types of bindery properties: ITEM and SET. SET properties are those that contain multiple bindery objects. (See "Introduction to Bindery Services" in this chapter.)

See Also

NWAddObjectToSet
NWDeleteObject
NWDeleteProperty

NWDeleteProperty

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function removes a property from a bindery object on the file server specified with the file server connection ID (serverConnID).

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
char        objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16      objectType;
char        propertyName[NWMAX_PROPERTY_NAME_LENGTH];

ccode=NWDeleteProperty( serverConnID, objectName, objectType,
    propertyName );
```

Input

serverConnID	Passes the file server connection ID.
objectName	Passes a pointer to the object name whose property is being deleted.
objectType	Passes the type of the object whose property is being deleted. (See Appendix A, Bindery Object Types.)
propertyName	Passes a pointer to the property name to be deleted.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xF1	Bindery Security
0xF6	No Property Delete
0xFB	No Such Property
0xFC	No Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The objectName and objectType must uniquely identify the bindery object and cannot contain wildcard characters.

The propertyName parameter may contain wildcards. All matching properties of the bindery object are deleted when the propertyName contains wildcard characters.

Notes

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

See Also

NWDeleteObjectFromSet

NWGetBinderyAccessLevel

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the access level of the currently logged-in client based on the file server specified with the file server connection ID (serverConnID).

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint8       binderyAccessLevel;
uint32      objectID;

ccode=NWGetBinderyAccessLevel( serverConnID, &binderyAccessLevel,
      &objectID );
```

Input

serverConnID	Passes the server connection ID.
binderyAccessLevel	Passes a pointer to the space allocated for the current station's security access mask.
objectID	Passes a pointer to the space allocated for the object ID of the current logged in entity.

Output

binderyAccessLevel	Receives the current station's security access mask.
objectID	Receives the object ID of the current logged in entity.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x96	Server Out Of Memory
0xF1	Bindery Security
0xFE	Directory Locked
0xFF	Hardware Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

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

The binderyAccessLevel parameter is a byte in which the low 4 bits (nibble) indicate read security and the high 4 bits indicate write security. Read security controls which objects and properties the workstation can find when it scans the bindery. Write security controls which objects and properties the workstation can modify. Table 2-2, above, summarizes the security values.

For example, a binderyAccessLevel of 0x11 indicates that the requesting workstation has successfully logged in to the file server and does not have security equivalence to SUPERVISOR. This client is allowed access to objects that have LOGGED or OBJECT read or write security.

Read Security:

0xn0 = NWBS_ANY_READ
0xn1 = NWBS_LOGGED_READ
0xn2 = NWBS_OBJECT_READ
0xn3 = NWBS_SUPER_READ
0xn4 = NWBS_BINDERY_READ

Write Security:

0x0n = NWBS_ANY_WRITE
0x1n = NWBS_LOGGED_WRITE
0x2n = NWBS_OBJECT_WRITE
0x3n = NWBS_SUPER_WRITE
0x4n = NWBS_BINDERY_WRITE

NWGetObjectID

NetWare 2.x	NetWare 3.x	NetWare for AVIION Systems
✓	✓	✓

This function looks up an object ID of the stated object name and object type in the bindery on the file server specified with the file server connection ID (serverConnID).

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
char        objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16      objectType;
uint32      objectID;

ccode=NWGetObjectID( serverConnID, objectName, objectType, &objectID );
```

Input

serverConnID	Passes the server connection ID.
objectName	Passes a pointer to the name of the object being searched for.
objectType	Passes the bindery type of the object being searched for. (See Appendix A, Bindery Object Types.)
objectID	Passes a pointer to the space allocated for the object ID.

Output

objectID	Receives the object ID.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x96	Server Out Of Memory
0xF0	Illegal Wildcard
0xFC	No Such Object
0xFE	Directory Locked

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

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

The objectName and objectType parameters must uniquely identify the bindery object and cannot contain wildcard characters.

Notes

The requesting process must be logged in to the file server and have read access to the bindery object for this call to be successful.

See Also

NWChangeObjectSecurity
NWCreateObject

NWGetObjectName

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the name and object type of a bindery object on the file server specified with the file server connection ID (serverConnID).

Synopsis

```
#include "nwapi.h"

uint16      ccode;
uint16      serverConnID;
uint32      objectID;
char        objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16      objectType;

ccode=NWGetObjectName( serverConnID, objectID, objectName,
        &objectType );
```

Input

serverConnID	Passes the server connection ID.
objectID	Passes the object ID.
objectName	Passes a pointer to the string allocated for the object name.
objectType	Passes a pointer to the space allocated for the object type (optional).

Output

objectName	Receives the object name.
objectType	Receives the object type (optional).

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x96	Server Out Of Memory
0xF1	Bindery Security
0xFC	No Such Object
0xFE	Directory Locked

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The requesting process must be logged in to the file server and have read access to the bindery object for this call to be successful.

See Also

NWChangeObjectSecurity
NWCreateObject
NWGetObjectID

NWIsObjectInSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function searches a property of type SET for a specified object.

Synopsis

```
#include "nwapi.h"

NWBoolean_ts    ccode;
uint16          serverConnID;
char            objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16          objectType;
char            propertyName[NWMAX_PROPERTY_NAME_LENGTH];
char            memberName[NWMAX_MEMBER_NAME_LENGTH];
uint16          memberType;

ccode=NWIsObjectInSet( serverConnID, objectName, objectType,
    propertyName, memberName, memberType );
```

Input

serverConnID	Passes the server connection ID.
objectName	Passes a pointer to the name of the object containing the property being searched.
objectType	Passes the object type of the object containing the property being searched. (See Appendix A, Bindery Object Types.)
propertyName	Passes a pointer to the property name being searched (property type SET).
memberName	Passes the name of the bindery object being searched for.
memberType	Passes the bindery type of the object being searched for. (See Appendix A, Bindery Object Types.)

Output

ccode	This function returns a 1 when searched for object is in set, or a 0 when it is not.

Return Values

1 Object was found in set.

0 Object was NOT found in set. One of the following error codes is placed in NWErrno:

0xFC	No Such Object
0xF1	Bindery Security
0xEC	No Such Set
0xFE	Directory Locked

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The objectName, objectType, and propertyName parameters must uniquely identify the property and cannot contain wildcard specifiers.

The memberName and memberType parameters must uniquely identify the bindery object and cannot contain wildcard specifiers. The property must be of type SET.

This function does not expand members of type GROUP in an attempt to locate a specific member. For example, assume the following bindery objects and properties exist:

Object	Property	Property Value
JOAN		
SECRETARIES	GROUP_MEMBERS	JOAN's object ID
EMPLOYEES	GROUP_MEMBERS	SECRETARIES' object ID

JOAN is not considered a member of EMPLOYEES because she is not explicitly listed in the EMPLOYEES' GROUP_MEMBERS property. In addition, the bindery does not check for recursive (direct or indirect) membership definitions.

Notes

Read access to the property is required for this call.

For properties of type ITEM, the application must use NWScanPropertyValue.

See Also

NWAddObjectToSet

NWIsObjectPasswordOK

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function verifies the password of a bindery object on the file server specified with the file server connection ID (serverConnID).

Synopsis

```
#include "nwapi.h"

NWBoolean_ts    ccode;
uint16          serverConnID;
char            objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16          objectType;
char            objectPassword[NWMAX_PASSWORD_LENGTH];

ccode=NWIsObjectPasswordOK( serverConnID, objectName, objectType,
    objectPassword );
```

Input

serverConnID	Passes the file server connection ID.
objectName	Passes a pointer to the name of the bindery object whose password is being verified.
objectType	Passes the type of the bindery object whose password is being verified. (See Appendix A, Bindery Object Types.)
objectPassword	Passes a pointer to the password to be verified.

Output

None.

Return Values

1 Password is OK.

0 Password is not OK. One of the following errors codes is placed in NWErrno:

0xC5	Login Lockout
0xF1	Bindery Security
0xFB	No Such Property
0xFC	No Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The objectName and objectType parameters must uniquely identify the bindery object and cannot contain wildcards.

A bindery object without a password property is different from a bindery object with a password property that has no value. A workstation is not allowed to log in to a file server as a bindery object that does not have a password property. However, a workstation can log in without a password if the bindery object has been given a password property that contains no value.

Notes

The requesting workstation does not have to be logged in to the file server to make this call.

See Also

NWLoginToServerPlatform

NWOpenBindery

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function reopens a file server bindery that has been closed by a call to NWCloseBindery.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;

ccode=NWOpenBindery( serverConnID );
```

Input

serverConnID Passes the server connection ID

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFF	Failure
0xFE	Directory Locked

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The bindery files are normally kept open and locked. Therefore, this function is required only after a NWCloseBindery call has been made.

Notes

Only SUPERVISOR or a bindery object that is security equivalent to SUPERVISOR can open the bindery.

See Also

NWCloseBindery

NWRenameObject

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function renames an object in the bindery.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      oldObjectName[NWMAX_OBJECT_NAME_LENGTH];
char      newObjectName[NWMAX_OBJECT_NAME_LENGTH];
uint16    objectType;

ccode=NWRenameObject( serverConnID, oldObjectName, newObjectName,
    objectType );
```

Input

serverConnID	Passes the server connection ID.
oldObjectName	Passes a pointer to the name of a currently defined object in the bindery.
newObjectName	Passes a pointer to the new object name.
objectType	Passes the object's bindery type. (See Appendix A, Bindery Object Types.)

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xEE	Object Exists
0xF0	Illegal Wildcard
0xF3	No Object Rename
0xNo	Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Notes

The oldObjectName, newObjectName, and ObjectType parameters must uniquely identify the bindery object and cannot contain wildcard specifiers. Only SUPERVISOR or a bindery object that is security equivalent to SUPERVISOR can rename bindery objects.

NWScanObject

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function searches for a bindery object name.

Synopsis

```
#include "nwapi.h"

NWBoolean_ts      ccode;
uint16            serverConnID;
char              searchObjectName[NWMAX_OBJECT_NAME_LENGTH];
uint16            searchObjectType;
int32             sequence;
NWObjectInfo_t    objects;

sequence=-1;
ccode=NWScanObject( serverConnID, searchObjectName, searchObjectType,
    &sequence, &objects );
```

Input

serverConnID	Passes the server connection ID.
searchObjectName	Passes a pointer to the object name to be searched for (wildcards: * or ?).
searchObjectType	Passes the object type to be searched for; wildcard value: 0xFFFF. (See Appendix A, Bindery Object Types.)
sequence	Passes a pointer to the space allocated for the object ID of the next matching object.
objects	Passes a pointer to the structure allocated for the found object information. (See Appendix A, NWObjectInfo_t Structure.)

Output

sequence	Receives the object ID of the next matching object.
objects	Receives the information on the found object. (See Appendix A, NWObjectInfo_t Structure.)

Return Values

1 Object was found.

0 Object was not found. One of the following error codes is placed in NWErrno:

0xEF	Illegal Name
0xFC	No More Objects
0x93	No Read Privileges
0xEF	Illegal Name

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function is used iteratively to scan the bindery for all objects that match both the searchObjectName and the searchObjectType parameters. The sequence parameter should be set to -1 for the first search. Upon return, sequence automatically receives a number to be used as the object identification for the next call.

The NWObjectInfo_t structure contains the following fields:

```
char    objectName[NWMAX_OBJECT_NAME_LENGTH];
uint32  objectID;
uint16  objectType;
uint8   objectState;
uint8   objectSecurity;
```

The objectState field receives one of the following flags (optional):

```
NWBF_STATIC  = matching object is static
NWBF_DYNAMIC = matching object is dynamic
```

The objectSecurity parameter is a byte in which the low 4 bits (nibble) control read security and the high 4 bits control write security. Read security determines which clients can find the bindery object when they scan for it. Write security defines which clients can create properties for the bindery object. Below is a chart that lists these security options.

When scanning several objects, the application scans until NWErrno equals No More Objects.

For example, a bindery object with an objectSecurity of 0x31 can be viewed by any client that has successfully logged in to the file server, but only clients with security equivalence to SUPERVISOR can add properties.

Read Security:

```
0xn0 = NWBS_ANY_READ
0xn1 = NWBS_LOGGED_READ
0xn2 = NWBS_OBJECT_READ
0xn3 = NWBS_SUPER_READ
0xn4 = NWBS_BINDERY_READ
```

Write Security:

```
0x0n = NWBS_ANY_WRITE
0x1n = NWBS_LOGGED_WRITE
0x2n = NWBS_OBJECT_WRITE
0x3n = NWBS_SUPER_WRITE
0x4n = NWBS_BINDERY_WRITE
```

Notes

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

NWScanProperty

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function searches for properties in a bindery object.

Synopsis

```
#include "nwapi.h"

NWBoolean_ts        ccode;
uint16              serverConnID;
char                objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16              objectType;
char                searchPropertyName[NWMAX_PROPERTY_NAME_LENGT
                    H];
int32               sequence;
NWPropertyInfo_t    property;
uint8               moreFlag

sequence=-1;
ccode=NWScanProperty( serverConnID, objectName, objectType,
    searchPropertyName, &sequence, &property, &moreFlag );
```

Input

serverConnID	Passes the server connection ID.
objectName	Passes a pointer to the name of the object whose properties are being scanned.
objectType	Passes the bindery type of the object containing the property. (See Appendix A, Bindery Object Types.)
searchPropertyName	Passes a pointer to the property name (with possible wildcards) being searched for. (See *NetWare® for AViiON® Series Systems C Interface Programmer's Guide*, Chapter 2, "NetWare Properties.")
sequence	Passes a pointer to the space allocated for the sequence number of the next matching object.
property	Passes a pointer to the structure allocated for the found property information. (See Appendix A, NWPropertyInfo_t Structure.)
moreFlag	Passes a pointer to the space allocated for an indicator of more properties found.

Output

sequence	Receives the sequence number of the next matching object.
property	Receives a structure containing information on the found property. (See Appendix A, NWPropertyInfo_t Structure.)

moreFlag — Receives the more properties flag:

0x00 = no more properties for this object
0xFF = more properties exist

Return Values

1 Successfully found a property.

0 A property could NOT be found. One of the following error codes is placed in NWErrno:

0xFB	No More Properties
0xF0	Illegal Wildcard
0xFC	No Such Object
0xF9	No Property Read

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function iteratively scans the given bindery object for properties that match the searchPropertyName parameter. The sequence parameter should be assigned a -1 for the first scan. When the call returns, the moreFlag parameter contains 0xFF if the matched property is not the last property, and the sequence parameter receives the number to use in the next call.

When scanning several properties, the application should scan until NWErrno is equal to No More Properties.

The objectName and objectType parameters must uniquely identify the bindery object and cannot contain wildcard specifiers.

The NWPropertyInfo_t structure contains the following fields:

```
char    propertyName[NWMAX_PROPERTY_NAME_LENGTH];
uint8   propertyStateAndType;
uint8   propertySecurity;
uint8   propertyHasAValue;
```

The propertyName field is the name of the bindery property.

The propertyStateAndType field indicates the state and type of the property:

NWBF_STATIC or NWBF_DYNAMIC
ORed with NWBF_ITEM or NWBF_SET

The propertySecurity field receives a byte in which the low 4 bits (nibble) control read security and the high 4 bits control write security.

For example, a property with propertySecurity of 0x31 can be viewed by any client that has successfully logged in to the file server, but only a client with security equivalence to SUPERVISOR can write to the property.

The propertyHasAValue field receives one of the following flags indicating whether the property has a value:

0x00 = property has no value
0xFF = property has a value

Notes

This function requires read access to the bindery object as well as the property.

See Also

NWScanObject
NWWritePropertyValue

NWScanPropertyValue

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
		✓

This function reads the property value of a bindery object.

Synopsis

```
#include "nwapi.h"

NWBoolean_ts        ccode;
uint16              serverConnID;
char                objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16              objectType;
char                propertyNa[NWMAX_PROPERTY_NAME_
                    LENGTH];
uint8               segmentNumber;
uint8               segmentData[NWMAX_SEGMENT_DATA_
                    LENGTH];
uint8               moreSegments,
uint8               propertyType;

segmentNumber=1;
ccode=NWScanPropertyValue( serverConnID, objectName, objectType,
        propertyName, &segmentNumber, segmentData, &moreSegments,
        &propertyType );
```

Input

serverConnID	Passes the server connection ID.
objectName	Passes a pointer to the object name containing the property.
objectType	Passes the object type of the object containing the property. (See Appendix A, Bindery Object Types.)
propertyName	Passes a pointer to the property name whose information is being retrieved.
segmentNumber	Passes a pointer to the segment number of the data to be read. (See description below.)
segmentData	Passes a pointer to the buffer allocated for the property data.
moreSegments	Passes a pointer to the space allocated for the "more segments" code: 0x00 = no more segments to be read; 0xFF = more segments to be read)
propertyType	Passes a pointer to the space allocated for the property type.

Output

segmentNumber	Receives an incremented number until no more segments are found.
segmentData	Receives the 128-byte buffer of property data. (See description below.)
moreSegments	Receives 0x00 if there are no more segments to be read; otherwise, it receives 0xFF
propertyType	Receives the property type. (See Appendix A, Bindery Property Types.)

Return Values

1 Object successfully found.

0 Object not found. One of the following error codes is placed in NWErrno:

0x93	No Read Privileges
0xEC	No Such Set
0xF9	No Property Read
0xFB	No Such Property

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function is used to iteratively read property values with more than 128 bytes of data.

The segmentNumber should be set to 1 to read the first data segment of a property and will be incremented for each subsequent call until the moreSegments flag is set to 0 or until call fails (ccode=0).

The objectName, objectType, and propertyName parameters must uniquely identify the property and cannot contain wildcard specifiers.

The propertyType indicates the type of data a property value contains. The SET property type indicates that the property's value contains a set of bindery object identifications. The bindery attaches no significance to the contents of a property value if the property is of type ITEM. If the property is of type SET, the data returned in segmentData is an array of bindery object IDs.

The bindery makes no attempt to coordinate activities among multiple stations that concurrently read or write data to a single property. This means that one station 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.

Notes

Read access to the property is required to successfully call this function.

See Also

NWCreateProperty

NWWritePropertyValue

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function writes the property value of a bindery object.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
char        objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16      objectType;
char        propertyName[NWMAX_PROPERTY_NAME_LENGTH];
uint8       segmentNumber;
char        dataBuffer[NWMAX_SEGMENT_DATA_LENGTH];
uint8       moreFlag;

segmentNumber=1;
ccode=NWWritePropertyValue( serverConnID, objectName, objectType,
        propertyName, segmentNumber, dataBuffer, moreFlag );
```

Input

serverConnID	Passes the server connection ID.
objectName	Passes a pointer to the affected object name.
objectType	Passes the object type. (See Appendix A, Bindery Object Types.)
propertyName	Passes a pointer to the property name (type ITEM).
segmentNumber	Passes the segment number of the written data (128-byte chunks.) See "Description" below.
dataBuffer	Passes a pointer to the 128-byte buffer that contains the data. (See "Description" below.)
moreFlag	Passes a flag indicating whether more segments are being written: 0x00 = no more data segments 0xFF = more data segments

Output

None.

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0xE8	Write to Group
0xF8	No Property Write
0xFB	No Such Property
0xFC	No Such Object

Note: See Appendix B for a complete list of NetWare errors.

Description

A property value is data that is assigned to a particular bindery property. For example, a user's password is saved as a property value for the PASSWORD property.

The objectName, objectType, and propertyName parameters must uniquely identify the property and must not contain wildcard characters. The objectName 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 segmentNumber parameter indicates which segment of data is being written and should be assigned a value of 1 for the first segment. To write property data to more than one segment (128 bytes), this function should be called iteratively. In addition, the moreFlag parameter must contain a value of 0xFF unless you are writing to the last data segment. To signal Netware that the last segment is being written, and all further segments can be truncated, assign the moreFlag parameter to 0x00.

We recommend that property values be kept to a single segment (128 bytes) to improve bindery efficiency.

For NetWare 2.x, create property value segments sequentially. In other words, before you create segment N, you must have created all segments from 1 to N-1. However, once all segments of a property value have been established, segments can then be written at random. If the segment data is longer than 128 bytes, it is truncated.

The bindery makes no attempt to coordinate activities among multiple workstations concurrently reading or writing data to a single property. This means that one workstation 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.

Notes

A client must have write access to the property to call this function.

The objectName, objectType, and propertyName parameters must uniquely identify the property and cannot contain wildcard specifiers.

For properties of type SET, the application should use NWAddObjectToSet.

See Also

NWScanPropertyValue

End of Chapter

Chapter 3
Connection Service APIs

Function Calls

This chapter describes the following Connection Service APIs.

Introduction to Connection Services

The Connection Service calls allow developers to establish and destroy logical connections to a NetWare file server (creating utilities similar to the Netware LOGIN, ATTACH, and LOGOUT utilities), and return status information about those connections. Connection Services enable applications to do the following:

- Log in or attach objects to file servers
- Log out or detach objects from file servers
- Return information about a connection
- Return a clientConnID or a serverConnID

Connection Information

Connection information must be maintained by both the server and the connected client. The file server maintains two related tables:

- The File Server Connection Table
- The Password Table

The number of entries allowed in the table depends upon which version of the operating system the file server is running. NetWare for AViiON Systems allows 250 entries. Each entry in the File Server Connection table contains the network address of a client. The corresponding entry in the Password table contains the bindery object ID of the object type that established the connection between that client and the file server. The file server identifies a connection (both the connected client and the object attached through that client) by the connection's position (1 to 250) in these tables. This connection is known to the client as the clientConnID.

The following information is maintained in order to maintain a client connection:

- serverConnID
- clientConnID

The serverConnID is a number which represents a server to a client. The serverConnID is returned to the client by the NWAttachToServerPlatform function call.

Potential DG/UX errors

The Connection Service APIs use two DG/UX interrupt signals: SIGPOLL and SIGALRM. These interrupts can cause your C calls that do kernel reads or writes to fail. If your call fails during a kernel read or write, complete the following:

1. Check the errno value. If the value is EINTR, the interrupts have caused the error.

2. Redo your read or write.

We suggest that your program check for this condition on all kernel reads and writes. The code below is an example of how you could check for the condition.

```
# include <errno.h>
...
    rvalue = read( fd, buf, cnt );
    while( rvalue == -1 ) {
        if( errno == EINTR ) {
            rvalue = read( fd, buf, cnt );
        }
        else {
            break;
        }
    }
...
```

The interrupts can only cause the error when you have a transport open. The transport is opened using the NWAttachToServerPlatform call. Use the NWCloseTransport call to close the transport. After closing the transport, you will not have this interrupt problem.

NWAttachToServerPlatform

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function attaches the default client to the named file server.

Synopsis

```
#include "nwapi.h"

int       ccode;
char      fileServerName[NWMAX_SERVER_NAME_LENGTH];
uint16    serverConnID;

ccode=NWAttachToServerPlatform( fileServerName, &serverConnID );
```

Input

fileServerName	Passes a pointer to the name of the target file server.
serverConnID	Passes a pointer to the space allocated for the file server connection ID.

Output

serverConnID	Receives the file server connection ID.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xF8	Already Attached to Server
0xF9	No Free Connection Slots
0xFC	Unknown File Server
0xFF	No Response From Server

Note: See Appendix C for an explanation of possible DG/UX errors. See Appendix B for a complete listing of possible NetWare errors.

Description

This function initializes and sets up a client connection to a server and allows the client to log in to the server by using the NWLoginToServerPlatform function. This function returns a file server connection ID (serverConnID) for the new connection and places the newly attached file server's connection information in the client's connection tables. After using this function, the client can now login to the file server as an object.

The fileServerName array should contain either the name of the file server to attach to or an * (asterisk). If an asterisk is contained in the fileServerName array, the application will attach to the nearest file server. The asterisk may be used for utilities which do not require logging in.

Notes

This function will automatically open a transport.

See Also

NWLoginToServerPlatform
NWCloseTransport

NWClearClientConnID

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓		✓

This function clears a client connection number on the file server.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint16    clientConnID;

ccode=NWClearClientConnID( serverConnID, clientConnID );
```

Input

serverConnID	Passes the file server connection ID
clientConnID	Passes the client connection number to be cleared

Output

None

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFC	No Such Object
0xEF	Illegal Name
0xC1	No Account Balance
0xC5	Login Lockout

Note: See Appendix B for a listing of possible NetWare errors.

Description

Clearing a connection will log a client off the network. The client must then re-attach and login again in order to establish a new connection.

The calling application must be logged in as supervisor or have equivalent rights.

See Also

NWGetObjectClientConnIDs

NWCloseTransport

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
		✓

This function closes the underlying transport protocol. This call is not necessary for the application to make if the operating system closes open devices on completion of processes.

Synopsis

```
#include "nwapi.h"

int ccode;

ccode=NWCloseTransport();
```

Input

None.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. The following error code is placed in NWErrno:

0x03 Transport Close Error

Note: See Appendix C for an explanation of possible DG/UX errors. See Appendix B for a complete listing of possible NetWare errors.

Description

This function closes a transport after the client has logged out of a file server and detached from all connections. This call should be made at the end of the application to close the underlying transport.

Notes

The client would not be able to establish or use any connections unless a transport is open. A transport is automatically opened with NWAttachToServerPlatform.

See Also

NWAttachToServerPlatform
NWDetachFromServerPlatform
NWLogoutFromServerPlatform

NWDetachFromServerPlatform

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function breaks a client-file server connection.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;

ccode=NWDetachFromServerPlatform( serverConnID );
```

Input

serverConnID	Passes the file server connection ID.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFF	Connection Does Not Exist
0xFC	Unknown File Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

Detaching from a file server is not the same as logging out from a file server. Detaching relinquishes the connection number the client was using and breaks the connection. Before the client can send further requests to that file server, it must be reattached. Logging out from a file server preserves the connection ID and allows the client to log in again without reattaching.

See Also

NWAttachToServerPlatform
NWLogoutFromServerPlatform

NWGetClientConnID

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the connection number that the requesting client uses to communicate with the server specified by serverConnID.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint16    clientConnID;

ccode=NWGetClientConnID( serverConnID, &clientConnID );
```

Input

serverConnID	Passes the file server connection ID.
clientConnID	Passes a pointer to the current client's connection number.

Output

clientConnID	Receives the requesting client's connection ID number.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFB	Invalid Parameters
0x04	Not Connected To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The clientConnID parameter is an index into the Connection Table maintained by the file server.

Notes

This call is the same as NWGetClientConnID (with the old function name)

See Also

NWAttachToServerPlatform
NWGetServerConnIDList

NWGetConnectionInformation

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	

This function allows you to get information about a file server connection.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint16      clientConnID;
char        clientObjectName[NWMAX_OBJECT_NAME_LENGTH];
uint16      clientObjectType;
uint32      clientObjectID;
uint8       clientLoginTime[NWMAX_LOGIN_TIME_LENGTH];

ccode=NWGetConnectionInformation( serverConnID, clientConnID,
    clientObjectName, &clientObjectType, &clientObjectID,
    clientLoginTime );
```

Input

serverConnID	Passes the file server connection ID
clientConnID	Passes the client connection ID.
clientObjectName	Passes a pointer to the space allocated for the client's object name.
clientObjectType	Passes a pointer to the space allocated for the client's object type.
clientObjectID	Passes a pointer to the space allocated for the client's object ID.
clientLoginTime	Passes a pointer to the space allocated for the client's login time.

Output

clientObjectName	Receives the client's object name.
clientObjectType	Receives the client's object type.
clientObjectID	Receives the client's object ID.
clientLoginTime	Receives the client's login time.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFF	No Response From Server
0xFE	Server Bindery Locked
0xFC	No Such Object

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The clientLoginTime is returned in an array of 7 uint8s. The array will be filled with the following:

1st uint8=year	(0 through 99; for example: 90=1990)
2nd uint8=month	(1 through 12)
3rd uint8=day	(1 through 31)
4th uint8=hour	(0 through 23)
5th uint8=minute	(0 through 59)
6th uint8=second	(0 through 59)
7th uint8=dayOfWeek	(0 through 6, 0 = Sunday)

NWGetInternetAddress

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	

This function gets the internetwork address of any client on the network.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint16      clientConnID;
uint8       internetAddress[ NWMAX_INTERNET_ADDRESS_LENGTH ];

ccode=NWGetInternetAddress( serverConnID, clientConnID,
        internetAddress );
```

Input

serverConnID	Passes the file server connection ID.
clientConnID	Passes the client's connection ID.
internetAddress	Passes a pointer to the space allocated for the internet address.

Output

internetAddress	Receives the internet address.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x01	Invalid Parameter Length
0x04	Not Connected To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

An internetwork address comprises the networkNumber, the physicalNodeAddress, and the socketNumber. The internetwork address uniquely identifies a client throughout an internetwork. This address can be used to send packets directly to the client. The physicalNodeAddress is the address of the client's LAN board.

NWGetObjectClientConnIDs

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns a list of server maintained client connection ID numbers for a specified logged-in object.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16    objectType;
uint16    numberOfConnections;
uint16    connectionList[n];
uint16    maxListElements;

ccode=NWGetObjectClientConnIDs( serverConnID, objectName, objectType,
    &numberOfConnections, connectionList, maxListElements );
```

Input

serverConnID	Passes the file server connection ID.
objectName	Passes a pointer to the bindery object name of the object whose file server connection numbers are being returned (wildcards not allowed).
objectType	Passes the bindery object type of the object whose file server connection numbers are being obtained. (See Appendix A, Bindery Object Types.)
numberOfConnections	Passes a pointer to the space allocated for the number of server connections found for the specified object.
connectionList	Passes a pointer to the array allocated for the object's server connection ID numbers.
maxListElements	Passes the number of connectionList elements that have been allocated (n).

Output

numberOfConnections	Receives the number of server connections found for the specified object.
connectionList	Receives the server connection numbers for the specified object.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFC	No Such Object
0xEF	Illegal Name
0xC1	No Account Balance
0xC5	Login Lockout

Note: See Appendix B for a listing of possible NetWare errors.

Description

You should allocate as many elements as you think necessary for the connectionList parameter and then put that amount in the maxListElements parameter. This function will return less than or equal the amount that the application specifies. If fewer clientConnIDs were found than were requested, only numberOfConnections amount will be copied into connectionList.

See Also

NWClearClientConnID

NWGetServerConnID

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the file server connection ID.

Synopsis

```
#include "nwapi.h"

int       ccode;
char      fileServerName[NWMAX_SERVER_NAME_LENGTH];
uint16    serverConnID;

ccode=NWGetServerConnID( fileServerName, &serverConnID );
```

Input

fileServerName	Passes a pointer to the name of the target file server.
serverConnID	Passes a pointer to the space allocated for the server connection ID.

Output

serverConnID	Receives the file server connection ID.

Return Values

0 Successful.
-1 Unsuccessful. An error code will be placed in NWErrno:

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function gets the server connection ID (serverConnID) which is used in most other function calls to specify the server connection the client will access. This call will only return a serverConnID if a connection has already been established.

See Also

NWAttachToServerPlatform
NWGetClientConnID

NWGetServerConnIDList

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
		✓

This function returns a list of all current server connection IDs for the requesting client.

Synopsis

```
int       ccode;
uint16    connectionListBuffer[NWMAX_CONNECTION_LIST_LENGTH];
uint16    bufferSize;
uint16    numberOfConnsReturned;

ccode=NWGetServerConnIDList( connectionListBuffer, bufferSize,
    &numberOfConnsReturned );
```

Input

connectionListBuffer	Passes a pointer to the buffer allocated for the connection list.
bufferSize	Passes the number of connectionListBuffer elements which were allocated.
numberOfConnsReturned	Passes a pointer to the buffer in which the number of connections is returned.

Output

connectionListBuffer	Receives the connection list.
numberOfConnsReturned	Receives the number of connections that were actually placed in the connectionListBuffer.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFB	Invalid Parameters
0x04	Not Connected To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The maximum array size is NWMAX_CONNECTION_LIST_LENGTH.

The returned connection list contains all the server connection ID numbers in use by a client. This function will not return any more connection numbers than the number you specify with the bufferSize parameter. This function allows the application to control how many connections it is aware of.

See Also

NWGetClientConnID
NWGetServerPlatformInformation

NWLoginToServerPlatform

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function logs an object in to a connected file server.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      name[NWMAX_OBJECT_NAME_LENGTH];
uint16    objectType;
char      password[NWMAX_PROPERTY_NAME_LENGTH];

ccode=NWLoginToServerPlatform( serverConnID, name, type, password );
```

Input

serverConnID	Passes the file server connection ID.
name	Passes the name of the client to be logged in.
type	Passes the client object type to be logged in. (See Appendix A, Bindery Object Types.)
password	Passes the object password.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFC	No Such Object
0xEF	Illegal Name
0xC1	No Account Balance
0xC5	Login Lockout
0xDE	Bad Password
0xDF	Old Password

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This call enables a client to identify itself to a file server and thereby be cleared for access to the network's file system and resources. The requesting client must first use NWAttachToServerPlatform to establish a connection with the specified file server before logging in to the server.

Notes

If the object does not have a password, a null should be passed in place of the password parameter. Attaching to a file server is not the same as logging in. A client attaches to a file server to obtain a connection number (clientConnID). The client can then log in to the file server using that connection number.

See Also

NWAttachToServerPlatform
NWGetClientConnID
NWGetServerConnID
NWLogoutFromServerPlatform

NWLogoutFromServerPlatform

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function logs out the requesting client from the specified server.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;

ccode=NWLogoutFromServerPlatform( serverConnID );
```

Input

serverConnID	Passes the file server connection ID.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFF	Hardware Error
0x96	Server Out Of Memory

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

NWLogoutFromServerPlatform removes the requesting client's clientConnID from the file server but does not relinquish the serverConnID, and therefore a connection still exist to that server.

Detaching from a file server relinquishes the connection ID and breaks the connection forcing the client to re-attach before sending any further requests to the server.

See Also

NWDetachFromServerPlatform
NWLoginToServerPlatform

NWRegisterTimeoutErrorFunction

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
		✓

This function calls an application supplied function that is called when a timeout occurs. Timeouts occur when the application does not receive an acknowledgement from the server platform.

Synopsis

```
#include "nwapi.h"

int     ccode;
int     func();
int     *ptr;
ptr=(int *)func;

ccode=NWRegisterTimeoutErrorFunction( ptr );
```

Input

ptr Passes a pointer to the function the client wants to call.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. An error codes will be placed in NWErrno:

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows an application to determine how a timeout is handled. A timeout will occur when the server goes down, a network board malfunctions, or the cabling between the client and the file server is broken.

For example, the application function could handle the timeout by performing one of the following:

- Exit the application.
- Display a message.
- Reset the retry count and send out another set of requests for the client.

Although the transport mechanism for the APIs does have a preset retry timeout and retry count, this function can be used to signal another round of retries before it times out by calling a client function that returns a 0. This function could be used in a loop with other APIs to keep them going until this function allows the API to timeout.

If the application supplied function returns a non-zero value, the connection will be disabled.

End of Chapter

Chapter 4
File Service APIs

Function Calls

This chapter describes the File Service APIs listed below. Those APIs which are not currently available on file servers running NetWare for AViiON Systems or NetWare 2.x are designated [3.x]. Those that are not supported on file servers running NetWare 2.x but are supported on file servers running NetWare 3.x and NetWare for AViiON Systems are designated [3.x & NFAS].

Introduction to File Services

NetWare file services provide a set of supplementary calls that enable applications to manipulate files, directories, volumes, trustees, and their associated information.

NetWare rights are checked before a client can perform any file service functions.

The NWPath_t structure

The following structure is used to specify the location of NetWare file or directory:

```
typedef struct {
        NWDirHandle_ts      dirHandle;
        uint16              serverConnID;
        char                *pathName;
} NWPath_t;
```

dirHandle	Represents the directory handle allocated by the client pointing to a particular place in the directory structure
serverConnID	Represents the file server which contains the file system being accessed
pathName	Indicates a pointer which points to a character string which the client must allocate and fill in with a path name

The NWPath_t structure can be used in one of the following ways:

1. The application can pass a 0 in the dirHandle field and then pass a full path (of the target directory or file) in the pathName field.

2. The application can pass a previously allocated directory handle in the dirHandle field (see NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle) and then can pass a path in the pathName field which is relative to the directory that the dirHandle points to.

3. The pathName may be null if the dirHandle already references the full path.

NWClearObjectVolRestriction

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function clears any volume restrictions placed on an object with NWSetObjectVolRestriction.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint16      volNum;
uint32      objectID;

ccode=NWClearObjectVolRestriction( serverConnID, volNum,
        objectID );
```

Input

serverConnID	Passes the file server connection ID.
volNum	Passes the volume number.
objectID	Passes the object ID of the object whose restrictions you want to clear.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Notes

The client must have security equivalence to SUPERVISOR.

See Also

NWGetVolNum
NWSetObjectVolRestriction

NWCloseFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function closes a file after it has been opened with NWOpenFile, NWCreateFile, or NWCreateNewFile.

Synopsis

```
#include "nwapi.h"

int                ccode;
uint16             serverConnID;
NWFileHandle_ta    fileHandle;

ccode=NWCloseFile( serverConnID, fileHandle );
```

Input

serverConnID	Passes the file server connection ID.
fileHandle	Passes a pointer to the array containing the file handle.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function closes a file after you have opened it with NWOpenFile, NWCreateFile, or NWCreateNewFile and then deallocates the associated file handle. The file handle value is gained through either the NWCreateFile, NWCreateNewFile or the NWOpenFile function.

See Also

NWCreateFile
NWCreateNewFile
NWOpenFile

NWCreateDir

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	

This function creates a NetWare directory on the server specified by the connection ID.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;
uint16          inheritedRightsMask;

ccode=NWCreateDir( &path, inheritedRightsMask );
```

Input

path	Passes a pointer to the structure containing the directory handle, file server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
inheritedRightsMask	Passes the inherited rights mask for the new directory. (See Appendix A, Trustee Rights and Inherited Rights Mask.)

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The dirHandle parameter in NWPath_t Structure can be zero if the pathName parameter contains the complete path of the new directory, including the volume name. This call will not accept wild card characters, and the requesting client must have the Create right in the directory that will become the parent directory.

This call will not sequentially create a string of directories; this call only creates the last directory provided in the NWPath_t structure provided by the client. This call differs from NWCreateFile in that a handle is not returned. To obtain a directory handle to this directory, you must use NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle.

See Also

NWDeleteDir

NWCreateFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function allows you to create a new file name and will overwrite an existing file of the same name.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;
uint32          fileAttributes;
NWFileHandle_ta fileHandle;

ccode=NWCreateFile( &path, fileAttributes, fileHandle );
```

Input

path	Passes a pointer to the NWPath_t structure containing the file server connection ID, the directory handle, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
fileAttributes	Passes the file attributes of the file to be created. (See Appendix A, File Attributes.)
fileHandle	Passes a pointer to the array allocated for the fileHandle.

Output

fileHandle	Receives the file handle for the created file.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function creates a new file by passing a file name and receiving a file handle. This function automatically opens a file allowing you to call NWWriteFile. This function will overwrite an existing file of the same name.

The function fails if the client does not have Create and Erase rights or if a file with the same name has already been created and that file has been flagged System, Hidden, or Delete Inhibit.

This function will not sequentially create a string of directories; this function will only create one file at a time. For example, in the path below, dir1 and dir2 must already have been created, or this call will fail:

volume:\dir1\dir2\filename

See Also

NWCreateNewFile

NWCreateNewFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function allows you to create a new file, but does not allow you to overwrite an existing file of the same name.

Synopsis

```
#include "nwapi.h"

int                 ccode;
NWPath_t            path;
uint32              fileAttributes;
NWFileHandle_ta     fileHandle;

ccode=NWCreateNewFile( &path, fileAttributes, fileHandle );
```

Input

path	Passes a pointer to the NWPath_t structure containing the file server connection ID, the directory handle, and the path name. (See Appendix A, NWPath_t Structure.)
fileAttributes	Passes the file attributes of the file to be created. (See Appendix A, File Attributes.)
fileHandle	Passes a pointer to the array allocated for the fileHandle.

Output

fileHandle	Receives the file handle for the newly created file.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	File Already Exists
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function creates a file name and file handle. This function also opens the file for writing with NWWriteFile.

This function fails if a file exists with the same name in the same directory or the client does not have the Create right in the parent directory. Use the NWCreateFile function if you want to overwrite files with the same name.

This function will not sequentially create a string of directories; this function will only create one file at a time. For example, in the path below, dir1 and dir2 must already have been created, or this call will fail:

volume:\dir1\dir2\filename

See Also

NWCloseFile
NWCreateFile

NWDeleteDir

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function deletes a NetWare directory.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;

ccode=NWDeleteDir( &path );
```

Input

path — Passes a pointer to the NWPath_t structure containing the file server connection ID, the directory handle, and the path name. (See Appendix A, NWPath_t Structure.)

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function fails if any of the following conditions exist:

- The directory does not exist.
- Files exist in the existing directory.
- Another client has a directory handle pointing to the directory.
- The client does not have the Erase right to the target directory.
- The directory has the Delete Inhibit attribute set.

This function will not delete the volume root directory.

If the function succeeds, the function automatically deallocates any directory handles.

See Also

NWCreateDir

NWDeleteFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function marks a file for deletion.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
uint8       searchAttributes;

ccode=NWDeleteFile( &path, searchAttributes );
```

Input

path — Passes a pointer to the NWPath_t structure containing the file server connection ID, the directory handle, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)

searchAttributes — Passes the search attributes for the file, or files, to be deleted. (See Appendix A, Search Attributes.)

0x00	None (normal files)
0x02	Hidden
0x04	System
0x06	Both

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function deletes a NetWare file, or group of files if wildcard characters are used by marking them for deletion and rendering them unviewable to the end user.

The searchAttributes parameter is used to include system and/or hidden files. In other words, if only the system bit is set in the searchAttributes parameter then all files will be affected except hidden files. If only the hidden bit is set, all files will be affected except system files. When neither the hidden nor the system bit is set (0x00), then only files that are not hidden, system, or both will be affected.

The function fails if a file has been flagged delete inhibit. However, if the appropriate search attributes are passed with NWDeleteFile, NWDeleteFile can delete files that have been flagged System or Hidden.

Notes

These files may be recovered with NWRecoverSalvageableFile unless one of the following conditions exist:

- The file server is running NetWare for AViiON Systems.
- The file server is low on disk space and the set time for saving a deleted file has passed. Under these conditions, the operating system will allow another client to overwrite the file.
- The Purge attribute has been set on the file(s) or the parent directory.
- The operating system has been configured to immediately purge all deleted files.

See Also

NWPurgeSalvageableFile
NWRecoverSalvageableFile

NWDeleteTrustee

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function removes a trustee from a directory's or file's trustee list.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
uint32      trusteeObjectID;

ccode=NWDeleteTrustee( &path, trusteeObjectID );
```

Input

path	Passes a pointer to the NWPath_t structure containing the file server connection ID, the directory handle, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
trusteeObjectID	Passes the trustee object ID of the trustee to be deleted.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server
0xFE	No Trustee Exists

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function revokes a trustee's rights in a specific directory or file. The requesting client must have the Access Control right to the directory or file to delete a trustee.

Deleting the explicit assignment of an object's trustee in a directory or file is not the same as assigning that object no rights in the directory. If no rights are assigned to a directory or file, the object inherits the rights it has in the parent directory minus those revoked with the directory's or file's Inherited Rights Mask.

Notes

The trusteeObjectID can be obtained by calling NWGetObjectID.

See Also

NWGetEntrysTrustees
NWSetTrustee

NWFileCopy

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function copies a file, or portion of a file, to another file on the same file server.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
NWFileHandle_ta sourceFileHandle;
NWFileHandle_ta destinationFileHandle;
uint32          sourceFileOffset;
uint32          destinationFileOffset;
uint32          numberOfBytesToCopy;
uint32          numberOfBytesCopied;

ccode=NWFileCopy( serverConnID, sourceFileHandle,
destinationFileHandle, sourceFileOffset, destinationFileOffset,
numberOfBytesToCopy, &numberOfBytesCopied);
```

Input

serverConnID	Passes the file server connection ID.
sourceFileHandle	Passes a pointer to the source file handle. (See "Description" below.)
destinationFileHandle	Passes a pointer to the destination file handle. (See "Description" below.)
sourceFileOffset	Passes the offset, in the source file, where the copying is to begin.
destinationFileOffset	Passes the offset, in the destination file, where the copying is to begin.
numberOfBytesToCopy	Passes the maximum number of bytes to copy.
numberOfBytesCopied	Passes a pointer to the space allocated for the number of bytes actually copied.

Output

numberOfBytesCopied	Returns a pointer to the number of bytes actually copied.

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The source and destination files must reside on the same file server. If they do not, the following error is returned:

NOT_SAME_CONNECTION

To copy the entire source file, specify a value that matches or exceeds the file size in the numberOfBytesToCopy parameter.

If the destination file is new, the numberOfBytesCopied parameter will return the size of the destination file; otherwise, the numberOfBytesCopied parameter will return the number of additional bytes added.

The sourceFileHandle should be obtained by calling NWOpenFile and passing NWOR_READ in the accessRights parameter. For this function to succeed, the client must have Read rights to the file.

The destinationFileHandle should be obtained by calling NWOpenFile and passing NWOR_WRITE in the accessRights parameter. For this function to succeed, the client must have Create rights in the parent directory.

Notes

This function only allows copying files on the same file server. If the client chooses to copy files across different file servers, the client must create the destination file (NWCreateFile), read from the source file (NWReadFile), and write to the destination file (NWWriteFile).

See Also

NWCloseFile
NWCreateFile
NWCreateNewFile
NWOpenFile
NWReadFile
NWWriteFile

NWGetDirEntryInfo

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	✓

This function provides information about a directory through the directory handle.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
NWDirHandle_ts  dirHandle;
NWDirEntryInfo_t dirInfo;

ccode=NWGetDirEntryInfo( serverConnID, dirHandle, &dirInfo );
```

Input

serverConnID	Passes the file server connection ID.
dirHandle	Passes the directory handle associated with the directory you are requesting information for.
dirInfo	Passes a pointer to the NWDirEntryInfo_t structure allocated for the directory entry information. (See Appendix A, NWDirEntryInfo_t Structure.)

Output

dirInfo	Receives the directory entry information. (See Appendix A, NWDirEntryInfo_t Structure.)

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Notes

This call is useful for obtaining information from the root directory.

The dirHandle parameter must be allocated using NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle.

See Also

NWScanDirEntryInfo
NWAllocTemporaryDirHandle
NWAllocPermanentDirHandle

NWGetDirRestriction

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function checks for a directory's level and available blocks.

Synopsis

```
#include "nwapi.h"

int                 ccode;
uint16              serverConnID;
NWDirHandle_ts      dirHandle;
uint8               numberOfEntries;
NWDirRestriction_t  restrictions[n];
uint16              maxListElements;

ccode=NWGetDirRestriction( serverConnID, dirHandle,
        &numberOfEntries, restrictions, maxListElements );
```

Input

serverConnID	Passes the file server connection ID.
dirHandle	Passes the directory handle of the directory to be scanned.
numberOfEntries	Passes a pointer to the space allocated for the number of entries.
restrictions	Passes a pointer to the array of structures allocated for the directory restrictions. (See Appendix A, NWDirRestriction_t Structure.)
maxListElements	Passes the maximum number of objects that you expect to have restrictions.

Output

numberOfEntries	Receives the number of entries actually copied into the restrictions parameter (0 - n).
restrictions	Receives the directory restrictions for each entry. (see Appendix A, NWDirRestriction_t Structure.)

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function scans for the amount of disk space assigned to all directories between the current directory (referenced by the dirHandle) and the root directory. To find the actual amount of space available to a directory, scan all the entries returned in the restriction array and use the smallest one.

All directories will have a value in the maxBlocks parameter (from the NWDirRestriction_t structure). The maxBlocks parameter will return one of the following:

0x7FFFFFFF	No restrictions have ever been set.
negative value	Restrictions were set but they have been cleared. (Use a zero in NWSetDirRestriction to clear restrictions.)
positive value	Restrictions are set, and the positive value is the maximum value.

To calculate the amount of space in use, simply subtract availableBlocks from maxBlocks.

Notes

You must allocate a dirHandle before you make this call.

See Also

NWGetDirEntryInfo
NWScanDirEntryInfo
NWSetDirRestriction

NWGetEffectiveRights

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the client's effective rights in the specified directory or file.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
uint16      effectiveRights;

ccode=NWGetEffectiveRights( &path, &effectiveRights );
```

Input

path	Passes a pointer to the NWPath_t structure containing the directory handle, the file server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
effectiveRights	Passes a pointer to the space allocated for the effective rights for the directory or file. (See Appendix A, Trustee Rights and Inherited Rights Mask.)

Output

effectiveRights	Receives the effective rights for the directory or file. (See Appendix A, Trustee Rights and Inherited Rights Mask.)

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

For NetWare 3.x and NetWare for AViiON Systems, the requesting workstation's effective rights are determined with the inherited rights mask of the directory (or file), the client's trustee assignments, and the trustee assignments of the groups the client belongs to.

- If the client has been granted a trustee assignment in a parent directory of the specified directory, the client's effective rights are the trustee rights of the parent directory minus any rights revoked by the specified directory's (or file's) inherited rights mask.
- If the client has been granted a trustee assignment to the specified directory (or file), the client's effective rights are the current trustee assignment.
- If the client belongs to a group, the group's effective rights are added to the client's effective rights.

Notes

For NetWare 2.x, the effective rights to a file are always the same as the effective rights in the parent directory.

See Also

NWParseFullPath
NWDeleteTrustee
NWGetEntrysTrustees
NWSetTrustee

NWGetEntrysTrustees

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function scans an entry (directory or file) for trustees.

Synopsis

```
#include "nwapi.h"

int                 ccode;
NWPath_t            path;
uint8               numberOfEntries;
NWTrusteeRights_t   trustees;
uint16              maxListElements;

ccode=NWGetEntrysTrustees( &path, &numberOfEntries, &trustees,
        maxListElements );
```

Input

path	Passes a pointer to the NWPath_t structure containing the directory handle, the server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
numberOfEntries	Passes a pointer to the space allocated for the number of entries found.
trustees	Passes a pointer to the space allocated for the trustees. (See Appendix A, NWTrusteeRights_t Structure.)
maxListElements	Passes the maximum number of objects that you expect to have trustee rights.

Output

numberOfEntries	Receives the number of entries copied into the trusteeRights parameter (0 - n).
trusteeRights	Receives the trustee objectIDs and their associated rights. (See Appendix A, NWTrusteeRights_t Structure.)

Return Values

0	Successful
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x9C	No Trustees
0xFE	Directory Locked
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function scans an entry for trustees and returns their objectID and trustee rights. For NetWare 3.x, this call may be made to directories or files, since trustees are assigned to files as well as directories. For NetWare 2.x, this call may be made only to directories.

The client must have Access Control rights to the directory or file.

See Also

NWDeleteTrustee
NWSetTrustee

NWGetFileAttributes

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns a specified file's attributes.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
uint8       searchAttributes;
uint32      fileAttributes;

ccode=NWGetFileAttributes( &path, searchAttributes, &fileAttributes );
```

Input

path — Passes a pointer to the NWPath_t structure containing the directory handle, the file server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)

searchAttributes — Passes the search attributes of the file you are seeking. (See Appendix A, Search Attributes.)

0x00	None (normal files)
0x02	Hidden
0x04	System
0x06	Both

fileAttributes — Passes a pointer to the space allocated for the file's attributes. (See Appendix A, File Attributes.)

Output

fileAttributes — Receives the file's attributes. (See Appendix A, File Attributes.)

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function requires File Scan rights to the file.

See Also

NWSetFileAttributes

NWGetNameSpaceInfo

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	✓

This function returns all name spaces and data stream information for the specified file server and volume.

Synopsis

```
#include "nwapi.h"

int                 ccode;
uint16              serverConnId;
uint8               volNum;
NWNameSpaceInfo_t   nameSpace;

ccode=NWGetNameSpaceInfo( serverConnID, volNum, &nameSpace );
```

Input

serverConnID	Passes the file server connection ID.
volNum	Passes the number of the volume associated with the name space.
nameSpace	Passes a pointer to the structure allocated for the name space information. (See Appendix A, NWNameSpaceInfo_t Structure.)

Output

nameSpace	Receives the name space information. (See Appendix A, NWNameSpaceInfo_t Structure.)

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Notes

The volNum can be obtained by calling NWGetVolNum.

See also

NWGetVolNum

NWGetObjectVolRestriction

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function gets the volume restrictions placed on a specified object (such as a user).

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint8       volNum;
uint32      objectID;
int32       restriction;
int32       inUse;

ccode=NWGetObjectVolRestriction( serverConnID, volNum, objectID,
        &restriction, &inUse );
```

Input

serverConnID Passes the file server connection ID.

volNum	Passes the volume number.
objectID	Passes the object ID number for which the restrictions are being checked.
restriction	Passes a pointer to the space allocated for the object's volume restrictions.
inUse	Passes a pointer to the space allocated for the amount of volume space currently used by the object.

Output

restriction	Receives the object's volume restrictions on volume usage.
inUse	Receives the current amount of volume usage by the object.

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function returns the amount of space restriction based on 4K blocks on a specified object as well as the current amount of space used by the object. If restriction value returned is equal to 0x40000000, there are no restrictions.

Clients can receive space restriction information about themselves, but a client must have security equivalence to SUPERVISOR to receive information about other objects.

Notes

The objectID can be obtained by calling NWGetObjectID.

The volNum can be obtained by calling NWGetVolNum.

See Also

NWGetVolNum
NWGetObjectID

NWGetVolInfoWithHandle

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns information about a volume based on a specified NetWare directory handle.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
NWDirHandle_ts  dirHandle;
NWVolUsage_t    volUsage;

ccode=NWGetVolInfoWithHandle( serverConnID, dirHandle, &volUsage );
```

Input

serverConnID Passes the file server connection ID.

dirHandle — Passes the directory handle.

volUsage — Passes a pointer to the structure allocated for the volume usage information. (See Appendix A, NWVolUsage_t Structure.)

Output

volUsage — Receives the volume usage information. (See Appendix A, NWVolUsage_t Structure.)

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Description

This function returns information based on a directory handle. The information is placed in the NWVolUsage_t Structure.

Note: If the call is successful, the volume is mounted.

The following fields in the structure will always return a 0: purgableBlocks, notYetPurgableBlocks, maxDirEntriesUsed, volNum, isCached, isHashed, and isMounted.

Use NWGetVolNum to receive a valid value for the volume number.

Your version of NetWare for AViiON Systems may not support this function. See the release notice that accompanied your shipment for specific compatibility restrictions.

For NetWare 3.x, use NWGetVolUsage to return valid values for the following fields: purgableBlocks and notYetPurgableBlocks.

For NetWare 2.x, use NWGetVolUsage to return valid values for the following fields: maxDirEntriesUsed, volNum, isCached, isHashed, and isMounted.

Notes

To obtain a directory handle, the application must call NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle.

See Also

NWAllocPermanentDirHandle
NWAllocTemporaryDirHandle
NWGetVolNum
NWGetVolUsage

NWGetVolName

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the name of the volume associated with the specified volume number.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint8       volNum;
char        volName[NWMAX_VOLUME_NAME_LENGTH];

ccode=NWGetVolName( serverConnID, volNum, volName );
```

Input

serverConnID Passes the file server connection ID.

volNum — Passes the volume number for which the volume name is being obtained.

volName — Passes a pointer to the space allocated for the volume name (16 characters).

Output

volName — Receives the volume name.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function returns a volume's name by passing a volume number. The volNum parameter identifies the volume on the file server's Volume table. The Volume table contains information about each volume on the file server.

The volName parameter is 16 bytes long (including a null-byte). A volume name can be from 1 to 16 characters long and cannot include spaces or the following characters:

*	(asterisk)
?	(question mark)
:	(colon)
/	(slash)
\	(backslash)

If a volume name is fewer than 16 characters long, the remaining characters in the volName parameter are null. If volName is 16 characters long, it is not null-terminated.

See Also

NWGetVolNum

NWGetVolNum

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the volume number based on the file server connection ID number and the volume name. This call fails if the volume does not exist or the volume is not mounted.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
char        volName[NWMAX_VOLUME_NAME_LENGTH];
uint8       volNum;

ccode=NWGetVolNum( serverConnID, volName, &volNum );
```

Input

serverConnID Passes the file server connection ID.

volName — Passes a pointer to the volume name. Do not include a colon with the volume name.

volNum — Passes a pointer to the space allocated for the volume number.

Output

volNum — Receives the volume number.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x98	Volume Does Not Exist
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function returns a volume's number based on the file server serverConnID and volume name.

The volume name cannot contain wildcards.

If the volNum parameter is between 0 and the maximum allowable volume number on the network, the call is successful and a zero is returned.

See Also

NWGetVolName

NWGetVolsObjectRestrictions

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function will scan a volume for any object restrictions.

Synopsis

```
#include "nwapi.h"

int                     ccode;
uint16                  serverConnID;
uint8                   volNum;
uint8                   numberOfEntries;
NWUserRestriction_t     restrictions[n];
uint16                  maxListElements;

ccode=NWGetVolsObjectRestrictions( serverConnID, volNum,
        &numberOfEntries, restrictions, maxListElements );
```

Input

serverConnID	Passes the file server connection ID.
volNum	Passes the volume number.
numberOfEntries	Passes a pointer to the space allocated for the number of entries.
restrictions	Passes a pointer to the array of structures allocated for the user restrictions. (See Appendix A, NWUserRestriction_t Structure.)
maxListElements	Passes the maximum number of objects that you expect to have restrictions.

Output

numberOfEntries	Receives the number of entries that were copied into the restrictions array (0 - n).
restrictions	Receives the user restrictions. (See Appendix A, NWUserRestriction_t Structure.)

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function returns a list of the object restrictions for a specified volume. All restrictions are in 4K blocks. A restriction may be zero.

Notes

The volNum can be obtained by calling NWGetVolNum.

The client must have security equivalence to SUPERVISOR.

See Also

NWGetVolNum
NWSetObjectVolRestriction

NWGetVolUsage

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function gives you information about what is available, and in use, on a certain volume.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
uint8           volNum;
NWVolUsage_t    volUsage;

ccode=NWGetVolUsage( serverConnID, volNum, &volUsage );
```

Input

serverConnID Passes the file server connection ID.

volNum — Passes the volume number of the volume being checked.

volUsage — Passes a pointer to the structure allocated for the volume usage information. (See Appendix A, NWVolUsage_t Structure.)

Output

volUsage — Receives the filled-in structure with the volume usage information. (See Appendix A, NWVolUsage_t Structure.)

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x98	Volume Does Not Exist (NetWare 3.x and PNW)
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The volNum parameter identifies the volume on the file server's volume table, which contains information about each volume on the file server. Use NWGetVolNum to obtain a volume number.

For NetWare 3.x, the following fields in the NWVolUsage_t Structure will always return a 0: maxDirEntriesUsed, isCached, isHashed, isRemovable, isMounted. Use NWGetVolInfoWithHandle to return a valid value for isRemovable. This call fails if the volume does not exist or the volume isn't mounted.

For NetWare 2.x, the following fields in the NWVolUsage_t Structure will always return a 0: purgableBlocks, notYetPurgableBlocks, and sectorsPerBlock. Use NWGetVolInfoWithHandle to return a valid value for sectorsPerBlock.

See Also

NWGetVolNum
NWGetVolInfoWithHandle

NWMoveEntry

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function allows you to move and rename a file or directory.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;
uint8           searchAttributes;
NWDirHandle_ts  newDirHandle;
char            newPathName[NWMAX_DIR_PATH_LENGTH];

ccode=NWMoveEntry( &path, searchAttributes, newDirHandle,
newPathName );
```

Input

path — Passes a pointer to the structure containing the directory handle, server connection ID, and a pointer to the path name of the source file or directory. (See Appendix A, NWPath_t Structure.)

searchAttributes — Passes the search attributes for hidden or system files or directories. (See "Description" below and Appendix A, Search Attributes.)

none	0x00
hidden	0x02
system	0x04
both	0x06
directories	0x10
files	0x20

newDirHandle — Passes the new directory handle of the destination file or directory. (See "Description" below.)

newPathName — Passes a pointer to the destination file or directory name (See "Description" below.)

Output

None.

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The path parameter specifies the source file or directory that the client wants to move. The client must have Erase rights to the source directory or file and Create rights in the destination directory.

The searchAttributes parameter must contain either the directories or files search attribute bit so that the file server knows whether a file or directory is being moved. The directories or files search attribute bit may be OR'ed with the other search attribute values, if you want to move system or hidden files and directories.

The newDirHandle parameter is the directory handle for the destination of the file or directory. The destination directory must be on the same file server and volume as the source directory. The newDirHandle parameter can contain a zero value if a full path is passed in the newPathName parameter. To use a value other than zero, the newDirHandle must be allocated using NWAllocPermanentDirHandle or NWAllocTemporaryDirHandle.

The newPathName parameter is the new name for the directory or file in its new destination. If zero is passed in the newDirHandle parameter, a full path can be specified in the newPathName parameter. If a value other than zero is passed in the newDirHandle parameter, the newPathName parameter should specify only the directory or file name.

See Also

NWAllocPermanentDirHandle
NWAllocTemporaryDirHandle
NWMoveFile
NWRenameDir

NWMoveFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function moves or renames a file.

Synopsis

```
#include "nwapi.h"

int                 ccode;
NWPath_t            path;
uint8               searchAttributes;
NWDirHandle_ts      destDirHandle;
char                destFileName[NWMAX_FILE_NAME_LENGTH];

ccode=NWMoveFile( &path, searchAttributes, destDirHandle,
destFileName );
```

Input

path — Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name of the source file. (See Appendix A, NWPath_t Structure.)

searchAttributes — Passes the search attributes for hidden or system files. (See Appendix A, Search Attributes.)

none	0x00
hidden	0x02
system	0x04
both	0x06

destDirHandle — Passes the directory handle of the destination directory. (See "Description" below.)

destFileName — Passes a pointer to the destination file name. (See "Description" below.)

Output

None.

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

You can use this function to rename a file by simply moving it to the same directory with a new file name. If you use this call to move a file, the destination directory must reside on the same server and volume as the source directory.

The destDirHandle may contain a zero value, if a full path is passed in the destFileName parameter. To obtain a directory handle, call NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle.

The searchAttributes parameter is used to include system and/or hidden files. In other words, if only the system bit is set in the searchAttributes parameter then all files will be affected except hidden files. If only the hidden bit is set, all files will be affected except system files. When neither the hidden nor the system bit is set (0x00), then only files that are not hidden, system, or both will be affected.

Notes

To move a file to a different server, the application must create a file on the target server (NWCreateFile or NWCreateNewFile) and then read from the source file (NWReadFile) and write to the destination file (NWWriteFile).

See also

NWAllocTemporaryDirHandle
NWAllocPermanentDirHandle
NWCreateFile
NWCreateNewFile
NWReadFile
NWWriteFile

NWOpenFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function opens a previously created file for reading or writing.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;
uint8           searchAttributes;
uint8           accessRights;
NWFileHandle_ta fileHandle;

ccode=NWOpenFile( &path, searchAttributes, accessRights, fileHandle );
```

Input

path

Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)

searchAttributes

Passes the search attributes for hidden or system files. (See Appendix A, Search Attributes.)

none	0x00
hidden	0x02
system	0x04
both	0x06

accessRights

Passes the access rights of the file to open. (See Appendix A, Open Access Rights.)

fileHandle

Passes a pointer to the space allocated for the file handle of the file to be opened.

Output

fileHandle

Receives the file handle of file to be opened.

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x94	Invalid Open Access Rights
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function opens a NetWare file and returns the file handle for reading or writing.

Notes

You should use NWCloseFile after completing the read or write to the file.

See Also

NWCloseFile
NWReadFile
NWWriteFile

NWPurgeSalvageableFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function permanently deletes files that have been erased but are still recoverable.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
int32       entryID;
ccode=NWPurgeSalvageableFile( &path, entryID );
```

Input

path	Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
entryID	Passes the entryID for the file to be purged. (See "Notes" below.)

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

When a file is deleted, it can still be recovered for a time period and still uses disk space. This function can be used

- To permanently delete erased but recoverable files.
- To free the disk space being used by deleted, but still recoverable, files.

This function purges one file previously marked for deletion. Use NWScanSalvageableFiles until you find a file you want to purge. Then call NWPurgeSalvageableFile, passing in the entryID that corresponds to the desired file. The entryID is obtained from NWScanSalvageableFiles.

See Also

NWRecoverSalvageableFile
NWScanSalvageableFiles

NWReadFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function allows you to read a file.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
NWFileHandle_ta fileHandle;
uint32          startingOffset;
uint32          bytesToRead;
uint32          bytesActuallyRead;
char            data[n];

ccode=NWReadFile( serverConnID, fileHandle, &startingOffset,
        bytesToRead, &bytesActuallyRead, data );
```

Input

serverConnID	Passes the file server connection ID.
fileHandle	Passes a pointer to the file handle.
startingOffset	Passes the address of the offset where the file read should begin.
bytesToRead	Passes the maximum number of bytes to be read (should not exceed n).
bytesActuallyRead	Passes a pointer to the space allocated for the actual number of bytes read.
data	Passes a pointer to the space allocated for the data being read.

Output

startingOffset	Receives the new offset (previous offset plus the number of bytes read).
bytesActuallyRead	Receives the actual number of bytes that were read (0 - n).
data	Receives the data that is read.

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function reads from a file that has been previously opened with NWCreateFile, NWCreateNewFile, or NWOpenFile.

See also

NWCloseFile
NWCreateFile
NWCreateNewFile
NWOpenFile
NWWriteFile

NWRecoverSalvageableFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function restores a deleted, but salvageable file.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
int32       entryID;
char        newFileName[NWMAX_FILE_NAME_LENGTH];

ccode=NWRecoverSalvageableFile( &path, entryID, newFileName );
```

Input

path	Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
entryID	Passes the entryID corresponding to the file. (See "Description" below.)
newFileName	Passes a pointer to the space allocated for the filename. This space contains the name of the file to be restored. (This name may be the same name as the salvageable file's, unless another file was created with the same name.)

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function restores one file previously marked for deletion. You should use NWScanSalvageableFiles until you find a file you want to salvage. Then call NWRecoverSalvageableFile, passing in the entryID that corresponds to the desired file. The entryID is obtained from NWScanSalvageableFiles.

Notes

If the client creates more than one file with the same name as an erased file, the function renames the erased files, replacing the last two characters of the file extension with 00. For example,

TEST.DAT	becomes	TEST.D00
TEST	becomes	TEST.00

See Also

NWPurgeSalvageableFile
NWScanSalvageableFiles

NWRenameDir

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function allows you to change the name of a directory.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;
char            newDirName[NWMAX_DIR_NAME_LENGTH];

ccode=NWRenameDir( &path, newDirName );
```

Input

path	Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
newDirName	Passes a pointer to the array allocated for the new directory name.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The newDirName parameter should only contain the directory's new name, not a path specification. Names longer than the DOS 8.3 will be truncated.

This function will rename, not move, a directory. To move a directory, see NWMoveEntry.

See Also

NWMoveEntry

NWScanDirEntryInfo

NetWare 2.x	NetWare 3.x	NetWare for AVilON Systems
✓	✓	✓

This function scans for directory entry information such as entry names, attributes, and creation (date and time), archive (date and time), last modification (date and time).

Synopsis

```
#include "nwapi.h"

NWBoolean_ts        ccode;
NWPath_t            path;
int32               entryID;
uint8               searchAttributes;
NWDirEntryInfo_t    dirInfo;

entryID=-1;
ccode=NWScanDirEntryInfo( &path, &entryID, searchAttributes,
&dirInfo )
```

Input

path — Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See "Description" and Appendix A, NWPath_t Structure.)

entryID — Passes a pointer to the entryID of the previously scanned directory. (See "Description".)

searchAttributes — Passes the search attributes. (See Appendix A, Search Attributes.)

none	0x00
hidden	0x02
system	0x04
both	0x06

dirInfo — Passes a pointer to the structure allocated for the directory entry information. (See Appendix A, NWDirEntryInfo_t Structure.)

Output

entryID — Receives the entryID of the current directory.

dirInfo — Receives the directory entry information. (See Appendix A, NWDirEntryInfo_t Structure.)

Return Values

1	Successful
0	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The application must provide a search string in the pathName field of the NWPath_t structure. Use the following examples to determine the search string you want to specify in the pathName field:

Scan all directories in the sys volume	sys:*
Scan all directories under sys:dir1	sys:\dir1*
Scan directories under dir1 beginning with t	sys:\dir1\t*
Scan information on dir2	sys:\dir1\dir2

The entryID parameter should pass in a -1 for the first scan; for subsequent calls, the entryID of the previously-scanned directory should be passed. This entryID only has meaning for the file server. The application should not have to manipulate this value.

Notes

For applications talking to 2.x servers, the following fields in the NWDirEntryInfo_t structure will contain valid values:

entryName
creationDateAndTime
ownerID
inheritedRightsMask

The remaining structure members will be zero-filled.

See Also

NWGetDirEntryInfo
NWSetDirEntryInfo

NWScanFileEntryInfo

NetWare 2.x	NetWare 3.x	NetWare for AVilON Systems
✓	✓	✓

This function returns information about a file such as owner, size, attributes, last access (date and time), and creation (date and time).

Synopsis

```
#include "nwapi.h"

NWBoolean_ts        ccode;
NWPath_t            path;
int32               entryID;
uint8               searchAttributes;
NWFileEntryInfo_t   fileEntryInfo;

entryID=-1;
ccode=NWScanFileEntryInfo( &path, &entryID, searchAttributes,
        &fileEntryInfo );
```

Input

path — Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See "Description" and Appendix A, NWPath_t Structure.)

entryID — Passes a pointer to the entryID of the previously-scanned file. (See "Description".)

searchAttributes — Passes the search attributes. (See Appendix A, Search Attributes.)

none	0x00
hidden	0x02
system	0x04
both	0x06

fileEntryInfo — Passes a pointer to the structure allocated for the file entry information. (See Appendix A, NWFileEntryInfo_t Structure.)

Output

entryID — Receives the sequence number of the current file.

fileEntryInfo — Receives the file entry information. (See Appendix A, NWFileEntryInfo_t Structure.)

Return Values

1	Successful.
0	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The application must provide a search string in the pathName field of the NWPath_t structure. Use the following examples to determine the search string you want to specify in the pathName field:

Scan all files in the sys:dir1 directory	sys:dir1*
Scan all files in the sys:dir1 directory beginning with t	sys:dir1\t*

The entryID parameter should pass in a -1 for the first scan; for subsequent calls, the entryID of the previously-scanned file should be passed. This entryID only has meaning for the file server. The application should not have to manipulate this value.

Notes

The fileSize field in the NWFileEntryInfo_t structure contains the logical file size.

For applications talking to NetWare 2.x file servers, the following fields in the NWFileEntryInfo_t structure will be zero:

archiverID
updatorID
inheritedRightsMask
nameSpaceID

See Also

NWSetFileEntryInfo

NWScanSalvageableFiles

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function returns information on deleted, but salvageable, files.

Synopsis

```
#include "nwapi.h"

NWBoolean_ts            ccode;
uint16                  serverConnID;
NWDirHandle_ts          dirHandle;
int32                   entryID;
NWSalvageableInfo_t     salvageInfo;

entryID=-1;
ccode=NWScanSalvageableFiles( serverConnID, dirHandle, &entryID,
        &salvageInfo );
```

Input

serverConnID	Passes the file server connection ID.
dirHandle	Passes the directory handle of the directory to be scanned.
entryID	Passes the entryID of the previously-scanned salvageable file. (See "Description".)
salvageInfo	Passes a pointer to the structure allocated for the salvageable entry information. (See Appendix A, NWSalvageableInfo_t Structure.)

Output

entryID	Receives the entryID of the current salvageable file. (See "Description".)
salvageInfo	Receives the structure allocated for the salvageable entry information. (See Appendix A, NWSalvageableInfo_t Structure.)

Return Values

1 Successful.
0 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The entryID parameter should pass in a -1 for the first scan; for subsequent calls, the entryID of the previously-scanned file should be passed. This entryID only has meaning for the file server. The application should not have to manipulate this value.

See Also

NWRecoverSalvageableFile
NWPurgeSalvageableFile

NWScanTrusteePaths

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the directory paths to which an object has trustee rights.

Synopsis

```
#include "nwapi.h"

NWBoolean_ts        ccode;
uint16              serverConnID;
uint32              objectID;
uint8               volNum;
int32               entryID;
uint16              trusteeAccessRights;
char                directoryPath[NWMAX_DIR_PATH_LENGTH];

entryID=-1;
ccode=NWScanTrusteePaths( serverConnID, objectID, volNum, &entryID,
        &trusteeAccessRights, directoryPath );
```

Input

serverConnID	Passes the server connection ID.
objectID	Passes the object ID of the user or group for which the trustee information is to be found.
volNum	Passes the volume number of the volume being searched.
entryID	Passes a pointer to the entryID of the previously-scanned directory path. (See "Description".)
trusteeAccessRights	Passes a pointer to the space allocated for the trustee's access mask. (See Appendix A, Trustee Rights and Inherited Rights Mask.)
directoryPath	Passes a pointer to the space allocated for the current trustee's directory path name.

Output

entryID	Receives the entryID of the current directory. (See "Description".)
trusteeAccessRights	Receives the trustee's access mask. (See Appendix A, Trustee Rights and Inherited Rights Mask.)
directoryPath	Receives the current trustee's directory path name.

Return Values

1 Successful.
0 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFC	No Such Object
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function is used iteratively to determine all of a bindery object's trustee directory paths and corresponding access masks.

The entryID parameter should pass in a -1 for the first scan; for subsequent calls, the entryID of the previously-scanned directory path should be passed. This entryID only has meaning for the file server. The application should not have to manipulate this value.

Notes

Only SUPERVISOR, the object, or a bindery object that is security equivalent to SUPERVISOR or the object, can scan an object's trustee directory paths.

The objectID can be obtained by calling NWGetObjectID.

The volNum can be obtained by using NWGetVolNum.

See Also

NWGetObjectID
NWGetVolNum

NWSetDirEntryInfo

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function sets or changes information kept about a directory such as owner, attributes, creation (date and time), or last access (date and time).

Synopsis

```
#include "nwapi.h"

int                 ccode;
NWPath_t            path;
uint8               searchAttributes;
uint32              changeAttributes;
NWDirEntryInfo_t    dirEntryInfo;

ccode=NWSetDirEntryInfo( &path, searchAttributes, changeAttributes,
        &dirEntryInfo);
```

Input

path — Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)

searchAttributes — Passes the search attributes for any hidden or system files. (See Appendix A, Search Attributes.)

none	0x00
hidden	0x02
system	0x04
both	0x06

changeAttributes — Passes the new attributes for the directory entry. (See Appendix A, Change Attributes.)

dirEntryInfo — Passes a pointer to the structure allocated for the directory entry information. (See Appendix A, NWDirEntryInfo_t Structure.)

Output

None.

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function sets information kept on directories. This information can be seen in the NWDirEntryInfo_t structure. A pointer to the entire structure should be passed, even if only one item is being changed. Furthermore, the change attributes must be passed which correspond to the data being changed. For example, if the owner of the directory is being changed, then the entire structure would be allocated and the new ownerID would be put in the ownerID field of the structure. Correspondingly, the NWCA_OWNER_ID change attribute would be passed in the changeAttributes parameter. (The ownerID is the object ID of the owner.)

If you use this function to change the name of a directory, you must change the path.path name to the new directory name for subsequent calls.

Notes

If more than one item is being changed, the change attributes may be OR'ed together. If you only want to change the directory's inherited rights mask, use NWSetDirsInheritedRightsMask.

For NetWare 2.x servers, the only data which can be changed (as referenced in the NWDirEntryInfo_t structure) is:

creationDateAndTime
ownerID
inheritedRightsMask

For NetWare 3.x and NetWare for AViiON Systems servers, the data that cannot be changed is: nameSpaceID

See Also

NWScanDirEntryInfo
NWSetDirsInheritedRightsMask

NWSetDirRestriction

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function sets (or clears) a directory's restrictions.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
NWDirHandle_ts  dirHandle;
int32       restriction;

ccode=NWSetDirRestriction( serverConnID, dirHandle, restriction );
```

Input

serverConnID	Passes the file server connection ID.
dirHandle	Passes the directory handle of the directory that will have its restrictions set.
restrictions	Passes the restrictions.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function requires that the application pass an allocated directory handle to the directory to which the restrictions apply. The directory handle can be obtained by calling either NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle.

The restriction parameter passes a 0 to clear all restrictions or a number corresponding to the space available in the directory. Restrictions are in 4K blocks; therefore, a restriction of 1 will restrict the space usage in a particular directory to 4K.

See Also

NWGetDirRestriction
NWAllocTemporaryDirHandle
NWAllocPermanentDirHandle

NWSetDirsInheritedRightsMask

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function sets the rights mask for a directory path.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
uint16      newRightsMask;

ccode=NWSetDirsInheritedRightsMask( &path, newRightsMask );
```

Input

path	Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
newRightsMask	Passes the rights that you want to grant the directory's rights mask. (See Appendix A, Trustee Rights and Inherited Rights Mask.)

Output

None.

Return Values

0	Successful.
-1	Unsuccessful.

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function modifies an inherited rights mask for a directory by replacing the existing mask. The function does not add or subtract rights from the existing mask. You pass all rights you want in the rights mask with the newRightsMask parameter.

See Also

NWGetDirEntryInfo
NWScanDirEntryInfo
NWSetFilesInheritedRightsMask

NWSetFileAttributes

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function modifies a file's attributes.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
uint8       searchAttributes;
uint32      fileAttributes;

ccode=NWSetFileAttributes( &path, searchAttributes, fileAttributes );
```

Input

path — Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)

searchAttributes — Passes the search attributes. (See Appendix A, Search Attributes.)

none	0x00
hidden	0x02
system	0x04
both	0x06

fileAttributes — Passes the file attributes to be set on the file designated in the pathName field in the NWPath_t structure. (See Appendix A, File Attributes.)

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The searchAttributes parameter is used to include system and/or hidden files. In other words, if only the system bit is set in the searchAttributes parameter then all files will be affected except hidden files. If only the hidden bit is set, all files will be affected except system files. When neither the hidden nor the system bit is set (0x00), then only files that are not hidden, system, or both will be affected.

See Also

NWSetFileEntryInfo
NWScanFileEntryInfo

NWSetFileEntryInfo

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function sets file information such as owner, creation (date and time) and last access (date and time).

Synopsis

```
#include "nwapi.h"

int                 ccode;
NWPath_t            path;
uint8               searchAttributes;
uint32              changeAttributes;
NWFileEntryInfo_t   fileInfo;

ccode=NWSetFileEntryInfo( &path, searchAttributes, changeAttributes,
        &fileInfo );
```

Input

path — Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)

searchAttributes — Passes the search attributes. (See Appendix A, Search Attributes.)

none	0x00
hidden	0x02
system	0x04
both	0x06

changeAttributes — Passes the change attributes. (See "Description" and Appendix A, Change Attributes.)

fileInfo — Passes a pointer to the structure allocated for the file information being set. (See Appendix A, NWFileEntryInfo_t Structure.)

Output

None.

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function sets information kept on files. This information can be seen in the NWFileEntryInfo_t structure. A pointer to the entire structure should be passed, even if only one item is being changed. Furthermore, the change attributes must be passed which correspond to the data being changed. For example, if the owner of the file is being changed, then the entire structure would be allocated and the new ownerID would be put in the ownerID field of the structure. Correspondingly, the NWCA_OWNER_ID change attribute would be passed in the changeAttributes parameter.

Notes

To only change the file attributes, use NWSetFileAttributes. To only change the file's inherited rights mask, use NWSetFilesInheritedRightsMask.

For all versions of NetWare servers, the following data cannot be changed:

fileSize
nameSpaceID

In addition, for NetWare 2.x, the following data cannot be changed:

entryName
archiverID
updatorID
updateDateAndTime
inheritedRightsMask

See Also

NWScanFileEntryInfo
NWSetFileAttributes
NWSetFilesInheritedRightsMask

NWSetFilesInheritedRightsMask

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function sets the rights mask for a file.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
uint16      newRightsMask;

ccode=NWSetFilesInheritedRightsMask( &path, newRightsMask );
```

Input

path	Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
newRightsMask	Passes the rights that you want to grant the file's rights mask. (See Appendix A, Trustee Rights and Inherited Rights Mask.)

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function modifies an inherited rights mask for a file by replacing the existing mask. The function does not add or subtract rights from the existing mask. You pass all rights you want in the rights mask with the newRightsMask parameter.

See Also

NWScanFileEntryInfo
NWSetFileAttributes
NWSetFileEntryInfo

NWSetObjectVolRestriction

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
	✓	

This function sets restrictions on objects in a volume.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint8       volNum;
uint32      objectID;
int32       restriction;

ccode=NWSetObjectVolRestriction( serverConnID, volNum, objectID,
        restriction );
```

Input

serverConnID	Passes the file server connection ID.
volNum	Passes the volume number.
objectID	Passes the bindery object ID of the object for which the restrictions are being set.
restriction	Passes the objects volume restrictions.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function is similar to NWSetDirRestriction in that a space restriction is set, but the restriction applies to a specific object rather than overall. Restrictions are set in 4K blocks.

Notes

The objectID can be obtained by calling NWGetObjectID.

The volNum can be obtained by calling NWGetVolNum.

The client must have security equivalence to SUPERVISOR.

See Also

NWGetVolNum
NWGetObjectID
NWClearObjectVolRestriction
NWGetObjectVolRestriction

NWSetTrustee

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function creates a trustee for a file or directory. It also can changes a current trustee's trustee rights.

Synopsis

```
#include "nwapi.h"

int         ccode;
NWPath_t    path;
uint32      trusteeObjectID;
uint16      trusteeRightsMask;

ccode=NWSetTrustee( &path, trusteeObjectID, trusteeRightsMask );
```

Input

path	Passes a pointer to the NWPath_t structure containing the directory handle, server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)
trusteeObjectID	Passes the bindery object ID of the trustee.
trusteeRightsMask	Passes the trustee rights mask. (See "Description" and Appendix A, Trustee Rights and Inherited Rights Mask.)

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function assigns a user with specific rights to a directory or file; the user may already be a trustee in the directory or may be a new trustee. Once assigned, the user is called a trustee of that directory. To take rights away from a trustee simply pass a trusteeRightsMask without those trustee rights bits set.

If the trustee (represented by the trusteeObjectID) is already a trustee for this directory, all current trustee assignments will be replaced with the new trusteeRightsMask. This call will not add the new trusteeRightsMask to the current trustee rights.

Notes

The trusteeRightsMask is a value that can be obtained by ORing together all of the desired trustee rights. (See Appendix A, Trustee Rights and Inherited Rights Mask).

The trusteeObjectID can be obtained using NWGetObjectID.

See Also

NWGetObjectID

NWWriteFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function allows you to write to a file.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
NWFileHandle_ta fileHandle;
uint32          startingOffset;
uint32          bytesToWrite;
char            data[n];

ccode=NWWriteFile( serverConnID, fileHandle, &startingOffset,
        bytesToWrite, data );
```

Input

serverConnID	Passes the file server connection ID.
fileHandle	Passes a pointer to the file handle.
startingOffset	Passes a pointer to the offset where the file write is supposed to begin.
bytesToWrite	Passes the number of bytes to write.
data	Passes a pointer to data being written.

Output

startingOffset	Receives the new offset (previous offset plus the number of bytes written).

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0xFF	No Files Found
0x84	No Create Privileges
0x9B	Bad Directory Handle
0x9E	Invalid Filename
0xF8	Not Attached To Server

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function will write to a NetWare file after the file has been created and opened.

Notes

You must first use NWCreateFile, NWCreateNewFile or NWOpenFile to get a file handle for the file to be written to. The file should be closed using NWCloseFile after it has been written to.

See Also

NWCloseFile
NWCreateFile
NWCreateNewFile
NWReadFile

End of Chapter

Chapter 5
Path Service APIs

Function Calls

This chapter describes the following Path Service APIs.

Introduction to Path Services

Path Services system calls provide developers with the necessary tools to do the following:

- Allocate directory handles
- Deallocate directory handles
- Return information about directory paths

Path Services system calls operate through a series of tables maintained by both the file server and the client. Clients must maintain tables which keep track of the directory handles they create. Directory handles represent a full directory path name and can be used as a convenient method for pointing to a particular place level in the directory structure. Most calls in the File System Services allow you to use directory handles when specifying a file or directory. The calls usually accept the directory handle in the NWPath_t structure.

The NWPath_t structure

The following structure is used to specify the location of NetWare file or directory:

```
typedef struct {
    NWDirHandle_ts      dirHandle;
    uint16              serverConnID;
    char                *pathName;
} NWPath_t;
```

dirHandle — Represents the directory handle allocated by the client pointing to a particular place in the directory structure

serverConnID	Represents the file server which contains the file system being accessed
pathName	Indicates a pointer which points to a character string which the client must allocate and fill in with a path name

The NWPath_t structure can be used in one of the following ways:

1. The application can pass a 0 in the dirHandle field and then pass a full path (of the target directory or file) in the pathName field.

2. The application can pass a previously allocated directory handle in the dirHandle field (see NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle) and then can pass a path in the pathName field which is relative to the directory that the dirHandle points to.

3. The pathName may be null if the dirHandle already references the full path.

NWAllocPermanentDirHandle

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function assigns a permanent directory handle.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;
uint16          driveNum;
NWDirHandle_ts  newDirHandle;
uint16          effectiveRightsMask;

ccode=NWAllocPermanentDirHandle( &path, driveNum,
    &newDirHandle,  &effectiveRightsMask );
```

Input

path	Passes a pointer to the NWPath_t structure created for the directory handle, the server connection ID, and a pointer to the directory path.
driveNum	Passes the drive number.
newDirHandle	Passes a pointer to the structure allocated for the new directory handle.
effectiveRightsMask	Passes a pointer to the space allocated for the client's effective rights to the directory connected via the newDirHandle parameter. See "Trustee rights and Inherited Rights Mask" in Appendix A.

Output

newDirHandle	Receives the new directory handle.
effectiveRightsMask	Receives the client's effective rights to the directory connected via the newDirHandle parameter. See "Trustee rights and Inherited Rights Mask" in Appendix A.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x98 Volume Does Not Exist
0x9C Invalid Path

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The directory handles allocated by this call are permanently allocated to the client's object ID by the file server unless deallocated by a call to NWDeallocateDirHandle, or overwritten by allocating over the same dirHandle. The client must keep track of dirHandles and the directories they represent for use with other function calls.

The driveNum parameter should be a unique number between 1-32. This number is required by the server when allocating dirHandles. The number could be used by the client in a table to represent a particular dirHandle, although the actual dirHandle must always be used when sending requests to the server.

The effectiveRightsMask receives the client's effective rights to the directory as determined by ANDing the directories inherited rights mask and the client's trustee rights in that directory.

Notes

If a volume is specified in the path name, the volume must be represented by the volume name followed by a colon. You should pass a different drive number for each dirHandle allocated to avoid overwriting an existing dirHandle.

See Also

NWAllocTemporaryDirHandle
NWDeallocateDirHandle
NWParseFullPath

NWAllocTemporaryDirHandle

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function assigns a temporary directory handle.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;
uint16          driveNum;
NWDirHandle_ts  newDirHandle;
uint16          effectiveRightsMask;

ccode=NWAllocTemporaryDirHandle( &path, driveNum,
    &newDirHandle, &effectiveRightsMask );
```

Input

path	Passes a pointer to the NWPath_t structure created for the directory handle, the server connection ID, and a pointer to the directory path.
driveNum	Passes the drive number.
newDirHandle	Passes a pointer to the structure allocated for the new directory handle.
effectiveRightsMask	Passes a pointer to the space allocated for the trustee's effective rights to the directory connected via the newDirHandle parameter. See "Trustee rights and Inherited Rights Mask" in Appendix A.

Output

newDirHandle	Receives the new directory handle.
effectiveRightsMask	Receives the client's effective rights to the directory connected via the newDirHandle parameter. See "Trustee rights and Inherited Rights Mask" in Appendix A.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x98 Volume Does Not Exist
0x9C Invalid Path

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The directory handles allocated by this call are automatically deallocated when either: the client logs out (or is somehow terminated), or a call is made to NWDeallocateDirHandle, or the dirHandle is overwritten by another allocation. The client should keep track of allocated directory handles and the directories they represent.

The driveNum parameter should be a unique number between 1-32. This number is required by the server when allocating dirHandles. The number could be used by the client in a table to represent a particular dirHandle, although the actual dirHandle must always be used when sending requests to the server.

The effectiveRightsMask receives the client's effective rights to the directory as determined by ANDing the directories inherited rights mask and the client's trustee rights in that directory.

Notes

If a volume is specified in the path name, the volume must be represented by the volume name followed by a colon. You should pass a different drive number for each dirHandle allocated to avoid overwriting an existing dirHandle.

See Also

NWAllocPermanentDirHandle
NWDeallocateDirHandle
NWParseFullPath

NWDeallocateDirHandle

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function deallocates an allocated directory handle.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
NWDirHandle_ts  dirHandle;

ccode=NWDeallocateDirHandle( serverConnID, dirHandle );
```

Input

serverConnID	Passes the file server connection ID.
dirHandle	Passes the directory handle to be deallocated.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful.

Note: See Appendix B for a complete listing of possible NetWare errors.

Notes

This API does not delete the directory handle. The API breaks the connection so that the directory handle does not point anywhere.

When a client terminates, logs out, or somehow loses it's connection, all directory handles for that workstation are deleted. An End-Of-Job also deallocates temporary directory handles.

See Also

NWAllocPermanentDirHandle
NWAllocTemporaryDirHandle
NWGetDirPath
NWParseFullPath

NWGetDirPath

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the path name of the directory to which the given directory handle is associated.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
NWDirHandle_ts  dirHandle;
char            dirPath[NWMAX_DIR_PATH_LENGTH];

ccode=NWGetDirPath( serverConnID, dirHandle, dirPath );
```

Input

serverConnID	Passes the file server connection ID.
dirHandle	Passes the directory handle for the directory whose path is to be reported.
dirPath	Passes a pointer to the space allocated for the directory path name associated with the directory handle (above).

Output

dirPath	Receives the directory path name associated with the directory handle (above).

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0x9B	Bad Directory Handle
0xFF	Path Not Locatable
0xFB	Invalid Parameters

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function returns the full path to the directory specified by the given directory handle.

See Also

NWAllocTemporaryDirHandle
NWParseFullPath

NWParseFullPath

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function parses a directory path string.

Synopsis

```
#include "nwapi.h"

int         ccode;
char        path[NWMAX_DIR_PATH_LENGTH];
char        server[NWMAX_SERVER_NAME_LENGTH];
uint16      serverConnID;
char        volName[NWMAX_VOLUME_NAME_LENGTH];
char        directories[NWMAX_DIR_PATH_LENGTH];

ccode=NWParseFullPath( path, server, &serverConnID, volName,
directories );
```

Input

path	Passes a pointer to the string containing the path to be parsed.
server	Passes a pointer to a string allocated for the server name.
serverConnID	Passes a pointer to structure containing the serverConnID.
volName	Passes a pointer to the string allocated for the name of the volume.
directories	Passes a pointer to the string allocated for the directory names.

Output

server	Receives the server name.
serverConnID	Receives the file server connection ID.
volName	Receives the name of the volume.
directories	Receives the directory names.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x9C	Invalid Path
0x9B	Bad Directory Handle
0xFF	Path Not Locatable
0xFB	Invalid Parameters

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The path parameter must contain a full path name.

If the path is on a local drive, an error is returned. If the path specifies a file server name and there are no connections to that file server, the NO_CONNECTIONS error is returned.

See Also

NWGetDirPath

NWSetDirHandle

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function sets the current directory for the specified directory handle.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;
NWDirHandle_ts  targetDirHandle;

ccode=NWSetDirHandle( &path, targetDirHandle );
```

Input

path	Passes a pointer to the NWPath_t structure created for the directory handle, the server connection ID, and a pointer to the directory path.
targetDirHandle	Passes the target directory handle that becomes the new directory handle for the specified directory.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful.

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function assigns the target directory handle to a directory path defined by the combined source directory handle and the source directory path.

The dirHandle from the NWPath_t structure is an index number from 1 to 255. The dirHandle points to a volume or a directory on the file server. A file server maintains a Directory Handle table for each logged in client. If this call fails, the targetDirHandle parameter remains unchanged.

This call can only change dirHandles among directories on the same file server. In cases where multiple file servers are being used, the dirHandle and targetDirHandle parameter must have the same server connection ID.

The pathName parameter from the NWPath_t structure can identify a full or partial directory path. A full directory path defines a volume or a directory on a given file server in the format VOLUME:DIRECTORY/.../DIRECTORY. A partial directory path specifies at least a directory, and possibly one or more parent directories.

Applications frequently combine a directory handle and a directory path to specify a target directory. For example, if the specified directory handle points to SYS: and the specified directory path = PUBLIC/WORDP, then the specified directory is SYS:PUBLIC/WORDP.

When an application defines a target directory using only a directory handle, the application must pass a null string in pathName parameter. When an application defines a directory using only a directory path, the application must set dirHandle to zero.

Notes

Directory handles must be kept track of separately for each file server connection.

See Also

NWParseFullPath
NWGetDirPath

End of Chapter

Chapter 6
Queue Management Service APIs

Function Calls

This chapter describes the following Queue Management Service APIs. Queue server specific calls are denoted by a bracketed S, [S].

Introduction to QMS

Queue Management Services (QMS) allow an application to create queues for controlling the flow of jobs and services on the network. A queue organizes client requests for a job server. A job server is software that resides at a specific workstation and provides services for other workstations on the network. Networks can have many different kinds of job servers, including print servers, archiving servers, compiling servers, message-sending servers, and so on. By placing requests into network queues, a job server can provide service that is both flexible and efficient.

Some of the function calls in this chapter would only be called by a queue server. Others might be called by user applications which submit queue jobs and maintain created queues.

NWAbortServicingQueueJob

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function signals the queue management software that a job can not be completed successfully.

Synopsis

```
#include "nwapi.h"

int                 ccode;
uint16              serverConnID;
uint32              queueID;
uint16              jobNumber;
NWFileHandle_ta     fileHandle;

ccode=NWAbortServicingQueueJob( serverConnID, queueID, jobNumber,
    fileHandle );
```

Input

serverConnID	Passes the job server connection ID.
queueID	Passes the bindery object ID for the queue in which the aborted job is located.
jobNumber	Passes the job number of the aborted job.
fileHandle	Passes a pointer to the file handle of the file associated with the aborted job.

Output

None.

Return Value

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: Because NWCloseFile function is called with NWAbortServicingQueueJob, you may receive an NWErrno = 0x001100xx. The 0x0011 indicates a file system services error. See Appendix B for a complete listing of possible NetWare errors.

Description

This function call allows a job server to inform the Queue Manager that it cannot complete servicing a job previously accepted for service. This function closes the job file and resets the job server's access rights to their original (login) values.

An aborted job returns to its former position in the job queue if its Service Restart flag (bit 0x10 of the jobControlFlags field in the NWQueueJobStruct_t) is set. For example, if a job is at the beginning of the queue before being called, it returns to the beginning of the queue after being aborted. An aborted job could, therefore, be next in line for service. For this reason, a job should not be aborted because of an error in the job's format or requests. Instead, use the NWFinishServicingQueueJob function.

Notes

A job should be aborted only if some temporary internal problem prevents it from completing. For example, a print job might be aborted if the printer has a paper jam. After the paper jam is corrected, the job server can service the job successfully.

If a job is attempting to access data without proper security clearance and is aborted, the job will remain in the queue and be serviced and aborted again and again. To remove a job from the job queue, a user would have to use the NWCloseFileAndAbortQueueJob call, or the queue server would have to use the NWFinishServicingQueueJob call.

Only a queue server that has previously accepted a job for service can make this function call.

See Also

NWChangeQueueJobEntry
NWCreateQueueFile
NWFinishServicingQueueJob
NWReadQueueJobEntry

NWAttachQueueServerToQueue

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function attaches the calling client to the specified queue as a queue server.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint32    queueID;

ccode=NWAttachQueueServerToQueue( serverConnID, queueID );
```

Input

serverConnID — Passes the file server connection ID.

queueID — Passes the bindery object ID of the queue being attached.

Output

None.

Return Value

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function call is created for queue job servers and must be used before the job server can perform any services in the queue. After the queue server has logged into the file server as a queue server (bindery object) this call establishes a connection between the queue server and the queue . If the queue server logs out of the file server, this connection to the queue will be detached.

Notes

A client must attach itself to a queue as a job server before it can service jobs from that queue. A queue can have as many as 25 job servers attached.The workstation making this function call must be security equivalent to one of the objects listed in the queue's Q_SERVERS group property.

NWChangeQueueJobEntry

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function changes the information about a job in a queue.

Synopsis

```
#include "nwapi.h";

int                 ccode;
uint16              serverConnID;
uint32              queueID;
NWQueueJobStruct_t  jobStruct;

ccode=NWChangeQueueJobEntry( serverConnID, queueID, &jobStruct );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue.
jobStruct	Passes a pointer to the job structure that contains the new information about the job. (See Appendix A, NWQueueJobStruct_t Structure.)

Output

None.

Return Value

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The following fields in the NWQueueJobStruct_t structure may be changed by the owner of the job or by a queue operator:

targetServerID
targetExecutionTime
jobType
jobControlFlags
jobDescription
queueRecord

If the caller is an operator, the Operator Hold flag can be reset to a value supplied by the caller.

Use NWChangeQueueJobPosition to change the job's service position in the queue.

Notes

The NWChangeQueueJobEntry function can be used in conjunction with the NWReadQueueJobEntry function to change a portion of the job's entry record. However, if the target entry is already being serviced, the NWChangeQueueJobEntry function returns a servicing error and makes no changes to the job's entry record.

If this call is being used in conjunction with printing and the NWPrintStruct_t, the structure must first be converted (using the NWConvertPrintStructToQueueStruct) before this call is made.

See Also

NWChangeQueueJobEntry
NWChangeQueueJobPosition
NWConvertPrintStructToQueueStruct
NWGetQueueJobList
NWReadQueueJobEntry
NWRemoveJobFromQueue

NWChangeQueueJobPosition

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function changes a job's position in a queue.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint32    queueID;
uint16    jobNumber;
uint8     newJobPosition;

ccode=NWChangeQueueJobPosition( serverConnID, queueID, jobNumber,
    newJobPosition );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the affected queue.
jobNumber	Passes the job number of the job being repositioned.
newJobPosition	Passes the job's new position.

Output

None.

Return Value

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The value of the newJobPosition parameter ranges from 1 to 250. Position 1 is the first position in the queue and position 250 is the last position in a full queue. If a specified position number places the job beyond the current end of the queue, the job is placed at the end of the current queue.

Notes

When a job is moved in the queue, the positions of all job entries are updated to reflect the change. Changing the position of a job being serviced has no effect on the service of that job. Be Aware that job positions change as other jobs in the queue are finished being serviced.

The application making this call must be logged in as supervisor.

See Also

NWChangeQueueJobEntry
NWGetQueueJobList
NWReadQueueJobEntry
NWRemoveJobFromQueue

NWChangeToClientRights

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function changes a queue server's current login identity to match the identity of the client for whom the queue server is acting.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint32    queueID;
uint16    jobNumber;

ccode=NWChangeToClientRights( serverConnID, queueID, jobNumber );
```

Input

serverConnID	Passes the queue server connection ID.
queueID	Passes the Bindery object ID of the queue.
jobNumber	Passes the job's job number.

Output

None.

Return Value

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows a queue server to change its current login identity to match the identity of the client for which it is acting. This is useful if the queue server must access files owned by the client but not submitted to the queue by the client (in other words, if the server must go out and retrieve files by itself). The queue server's login user ID and associated security equivalence list are replaced by the ID and security equivalence list of the user who placed the job in the queue.

This function does not change any path mappings that the queue server may have on the job server. However, all access rights to those directories are recalculated to conform to the rights of the queue client. Files opened before this call is made will continue to be accessible with the server's rights. Files opened after this call is made will be accessible only with the client's rights.

Notes

The job server is responsible for creating any path mappings that it may need to carry out the client's requests after this call has been made.

The NWRestoreQueueServerRights function reverses the effects of the NWChangeToClientRights function. In addition, the server's rights are automatically reset if the server issues a NWFinishServicingQueueJob or NWAbortServicingQueueJob function.

Only a queue server that has previously accepted a job for service can call this function.

See Also

NWAbortServicingQueueJob
NWFinishServicingQueueJob
NWRestoreQueueServerRights

NWCloseFileAndAbortQueueJob

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function signals the QMS that a job has not been created properly and should be removed from the queue.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
uint32          queueID;
uint16          jobNumber;
NWFileHandle_ta fileHandle;

ccode=NWCloseFileAndAbortQueueJob( serverConnID, queueID, jobNumber,
    fileHandle );
```

Input

serverConnID	Passes the queue server connection ID.
queueID	Passes the bindery object ID of the affected queue.
jobNumber	Passes the job entry number of the job whose service is being aborted.
fileHandle	Passes a pointer to the file handle of the aborted job's file (returned from the NWCreateQueueFile function call).

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0xD1	No Queue
0xD3	No Queue Rights
0xD4	Queue Full
0xD5	No Queue Job
0xF5	No Such Object
0xD6	No Job Right
0xD7	Queue Servicing
0xD8	Queue Not Active
0xFF	Invalid File Handle
0x30	Invalid Connection ID

Note: Because this API uses NWCloseFile, it is possible to get an NWErrno = 0x001100xx. The 0x0011 signifies a file system error. See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows the client to close a queue job and abort it. The jobNumber parameter contains the job number returned by QMS when the job was originally entered in the queue. The file associated with that job number is closed, and the job is deleted from the queue.

Notes

Only the client that created the queue job can call this function.

See Also

NWCloseFileAndStartQueueJob
NWCreateQueueFile
NWRemoveJobFromQueue

NWCloseFileAndStartQueueJob

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function closes a queue file and marks it ready for execution.

Synopsis

```
#include "nwapi.h"

int                ccode;
uint16             serverConnID;
uint32             queueID;
uint16             jobNumber;
NWFileHandle_ta    fileHandle;

ccode=NWCloseFileAndStartQueueJob( serverConnID, queueID, jobNumber,
    fileHandle );
```

Input

serverConnID	Passes the queue server connection ID.
queueID	Passes the bindery object ID of the queue in which the specified job was placed.
jobNumber	Passes the job number of the job to be serviced.
fileHandle	Passes a pointer to the file handle of the file associated with the job to be executed (returned from the NWCreateQueueFile function call).

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0xD1	No Queue
0xD3	No Queue Rights
0xD5	No Queue Job
0xF5	No Such Object
0xD7	Queue Servicing
0xD8	Queue Not Active
0xFF	Invalid File Handle
0x30	Invalid Connection ID

Note: Because this API uses NWCloseFile, it is possible to get an NWErrno = 0x001100xx. The 0x0011 signifies a file system error. See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows the workstation to close a queue job file and mark the job for execution.

The jobNumber parameter contains the job number returned by QMS when the job was originally entered in the queue.

When this function finishes, the specified job is ready for execution, if the userHoldFlag and operatorHoldFlag fields are both cleared and the targetExecutionTime was either not specified or has elapsed (set with the jobStruct parameter when the file was created).

Notes

Only the client that created the job can call this function.

See Also

NWCloseFileAndAbortQueueJob
NWCreateQueueFile
NWRemoveJobFromQueue

NWCreateQueue

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function creates a new queue in the bindery and file system of the specified file server.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
char            queueName[NWMAX_QUEUE_NAME_LENGTH];
uint16          queueObjectType;
NWDirHandle_ts  directoryHandle;
char            queueSubdirectory[NWMAX_QUEUE_SUBDIR_LENGTH];
uint32          newQueueID;

ccode=NWCreateQueue( serverConnID, queueName, queueObjectType,
    directoryHandle, queueSubdirectory, &newQueueID );
```

Input

serverConnID	Passes the file server connection ID.
queueName	Passes a pointer to the name of queue to be created (48 characters).
queueObjectType	Passes a number indicating the bindery object type for the new queue.
directoryHandle	Passes the NetWare directory handle pointing to the directory in which the queue's property is to be created (0 if the queueSubdirectory parameter contains the full path).
queueSubdirectory	Passes a pointer to the absolute path or a path relative to the NetWare directory handle that will contain the queue files (119 characters, stored in the Q_DIRECTORY property).
newQueueID	Passes a pointer to the space allocated for the new queue ID number.

Output

newQueueID	Receives the new queue ID number.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99	Directory Full
0xFF	Failure
0xF5	No Object Create Privilege
0x30	Invalid Connection ID
0x9B	Invalid Dir Handle
0x98	Volume Does Not Exist
0xEE	Queue Exists

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function creates a queue in the bindery, using the type and name specified by the queueObjectType and queueName parameters. The following bindery object types are defined for queues:

NWOT_PRINT_QUEUE
NWOT_ARCHIVE_QUEUE
NWOT_JOB_QUEUE

This function also creates the Q_DIRECTORY property. The value for the Q_DIRECTORY property is determined by combining the directoryHandle and queueSubdirectory parameters.

QMS will use the directory handle and directory path parameters to create a queue directory that holds the system files containing the queue itself and the job files related to the queue entries. The directory path SYS:SYSTEM is commonly used for the queue directory. QMS uses this directory to store queue files until they are serviced.

Next, this function creates the following group properties:

- Q_SERVERS
- Q_OPERATORS
- Q_USERS

Notes

Only SUPERVISOR or a bindery object that is security equivalent to SUPERVISOR can create a queue.

See Also

NWDestroyQueue

NWCreateQueueFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function creates a queue file.

Synopsis

```
#include "nwapi.h"

int                     ccode;
uint16                  serverConnID;
uint32                  queueID;
NWQueueJobStruct_t      jobStruct;
NWFileHandle_ta         fileHandle;

ccode=NWCreateQueueFile( serverConnID, queueID, &jobStruct, fileHandle );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery's object ID for the queue.
jobStruct	Passes a pointer to the structure in which the information about the job is stored. (See Appendix A, NWQueueJobStruct _t Structure.)
fileHandle	Passes a pointer to the file handle of the file to be created in the queue.

Output

jobStruct	Receives the completed job structure. (See Appendix A, NWQueueJobStruct_t Structure.)
fileHandle	Receives the file handle of the job's associated file. (This file contains data pertaining to the job; for example, a print job file would contain the actual data to be printed.)

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99	Directory Full
0xD7	Queue Servicing
0xD0	Queue Error
0xD8	Queue Not Active
0xD1	No Queue
0xDA	Queue Halted
0xD3	No Queue Rights
0xFC	No Such Object
0xD4	Queue Full
0xFF	Failure
0x30	Invalid Connection ID

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows a client to enter a new job in a queue.

The following fields within the NWQueueJobStruct_t structure must be assigned values before this call can be made. (See Appendix A, NWQueueJobStruct_t Structure.)

targetServerID — The objectID of the queue server or 0xFFFFFFFF for any server. The following example assigns 0xB0000012 as the targetServerID:

```
jobStruct.targetServerID = 0xB0000012;
```

targetExecutionTime — The time you want the file processed. The field is a 6-byte field with the following format: year, month, day, hour, minute, second. Use 0xFFFFFFFFFFFF for first opportunity. The following example assigns November 1, 1991, 9:30:10 am as the targetExecutionTime:

```
jobStruct.targetExecutionTime[0] = 91;
jobStruct.targetExecutionTime[1] = 11;
jobStruct.targetExecutionTime[2] = 1;
jobStruct.targetExecutionTime[3] = 9;
jobStruct.targetExecutionTime[4] = 30;
jobStruct.targetExecutionTime[5] = 10;
```

jobType — The number representing the type of job serviced by the server; this number is server dependent. The following example assigns 0x00 as the jobType (0x00 means the queue server does not use this field):

```
jobStruct.jobType = 0;
```

jobControlFlags — The control flag that has been assigned to the job. Use any of the following:

```
NWCF_OPERATOR_HOLD
NWCF_USER_HOLD
NWCF_ENTRY_OPEN
NWCF_SERVICE_RESTART
NWCF_SERVICE_AUTO_START
```

The following example assigns NWCF_SERVICE_RESTART:

```
jobStruct.jobControlFlags = NWCF_SERVICE_RESTART;
```

jobDescription — A string containing the content or purpose of the job. The following example assigns "Print Job" as the job description:

```
strcpy(jobStruct.jobDescription, "Print Job");
```

The queueRecord field may need to be filled in.

- If the file being submitted to the queue is a NetWare print job, the client must first allocate a printRecord and fill in the NWPrintRecord_t Structure.

  Use NWConvertPrintStructToQueueStruct to fill in the queueRecord with the printRecord information. Then NWCreateQueueFile can be called.

  If the client wants to verify the printRecord after making this call, use NWConvertQueueStructToPrintStruct to convert the queueRecord field back in to the printRecord.

- If the file being submitted to the queue in not a NetWare print job and the queue server uses the queueRecord parameter, the queue server must provide its own function to fill in the queueRecord parameter.

- If the queue server does not use the queueRecord parameter, the parameter does not need to be filled in.

The file server fills in all other fields within the jobStruct parameter and returns it to the requesting client.

The job will not be serviced until the file is closed with NWCloseFileAndStartQueueJob.

Notes

This function can be used in conjunction with the NWReadQueueJobEntry function to change a portion of the job's entry record. However, if the target entry is already being serviced, NWChangeQueueJobEntry returns a Q_SERVICING error and makes no changes to the job's entry record.

See Also

NWChangeQueueJobEntry
NWCloseFileAndAbortQueueJob
NWCloseFileAndStartQueueJob
NWConvertPrintStructToQueueStruct
NWConvertQueueStructToPrintStruct
NWRemoveJobFromQueue

NWDestroyQueue

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function deletes a queue.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint32    queueID;

ccode=NWDestroyQueue( serverConnID, queueID );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue to be deleted.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0xFC	No Such Object
0xF4	No Object Delete Privilege
0x30	Invalid Connection ID
0xFF	Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function destroys the queue specified by the queueID parameter. All active jobs are aborted, all servers are detached from the queue, and all jobs in the queue are destroyed and their associated files deleted. The queue object and its associated properties are removed from the bindery and the queue's subdirectory is deleted.

Notes

Only SUPERVISOR or a bindery object that is security equivalent to SUPERVISOR can destroy a queue.

See Also

NWCreateQueue

NWDetachQueueServerFromQueue

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function removes the calling client from the queue's list of active queue servers.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint32    queueID;

ccode=NWDetachQueueServerFromQueue( serverConnID, queueID );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue from which the calling station is being detached.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function removes the requesting client from the queue's list of active queue servers. If the requesting client is servicing a job, that service is automatically aborted.

Notes

Only a workstation previously attached to the queue as a queue server can call this function.

See Also

NWAttachQueueServerToQueue
NWReadQueueServerCurrentStatus
NWSetQueueServerCurrentStatus

NWFinishServicingQueueJob

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function signals that a job has been completed successfully.

Synopsis

```
#include "nwapi.h"

int                 ccode;
uint16              serverConnID;
uint32              queueID;
uint16              jobNumber;
NWFileHandle_ta     fileHandle;

ccode=NWFinishServicingQueueJob( serverConnID, queueID, jobNumber,
    fileHandle );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue containing the job being finished.
jobNumber	Passes the job number of the job being finished.
fileHandle	Passes a pointer to the file handle for the file associated with the queue job.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: Because this call uses NWCloseFile, it is possible to get NWErrno=0x001100xx. The 0x0011 indicates a file system error. See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows a queue server to signal the QMS that it has serviced a job successfully. The job entry is then destroyed, and the job file is closed and deleted.

The calling queue server's access rights to the queue server are restored to their original (login) values.

Notes

Only a queue server that has accepted a job to service can call this function.

See Also

NWAbortServicingQueueJob
NWChangeToClientRights
NWServiceQueueJob

NWGetQueueJobFileSize

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the file size of the specified queue job.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint32      queueID;
uint16      jobNumber;
uint32      fileSize;

ccode=NWGetQueueJobFileSize( serverConnID, queueID, jobNumber,
    &fileSize );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue to which the job is associated.
jobNumber	Passes the number of the job for which the information will be obtained.
fileSize	Passes a pointer to the space allocated for the file size.

Output

fileSize	Receives the file size.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

NWGetQueueJobList

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns a list of all jobs associated with a given queue.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint32      queueID;
uint16      numberOfJobsInQueue;
uint16      listOfJobNumbers[NWMAX_NUMBER_OF_JOB_NUMBERS];

ccode=NWGetQueueJobList( serverConnID, queueID,
    &numberOfJobsInQueue, listOfJobNumbers );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue whose job list is being reported.
numberOfJobsInQueue	Passes a pointer to the space allocated for the number of jobs in the queue.
listOfJobNumbers	Passes a pointer to the array allocated for the job numbers.

Output

numberOfJobsInQueue	Receives the number of jobs currently in the Queue (0 - 250).
listOfJobNumbers	Receives the job numbers of all the jobs in the queue (0 - 250 numbers possible).

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full

0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows a program to get a list of all the jobs currently in a queue. When used in conjunction with the NWReadQueueJobEntry function, this function allows an application to retrieve information about all the jobs in a given queue. Because the QMS environment is multithreaded, however, the positioning, number and type of jobs in the queue can change between consecutive calls.

This function allows a workstation to determine how many jobs are in the queue at a particular instant and the job number of each. If a subsequent call to read information about a job in the queue fails with a NO_Q_JOB error, the requesting workstation can assume that either the job was deleted from the queue or its service was completed.

Notes

The workstation making this call must be security equivalent to one of the objects listed in the queue's Q_USERS or Q_OPERATORS group properties.

See Also

NWChangeQueueJobEntry
NWChangeQueueJobPosition
NWReadQueueJobEntry

NWReadQueueCurrentStatus

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the current status of a queue.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint32      queueID;
uint8       queueStatus;
uint16      numberOfJobsInQueue;
uint16      numberOfServers;
uint32      serverObjectIDList[NWMAX_NUMBER_OF_SERVER_
            OBJECT_IDS];
uint16      clientConnIDList[NWMAX_NUMBER_OF_SERVER_CONN_
                NUMBERS];

ccode=NWReadQueueCurrentStatus( serverConnID, queueID,
    &queueStatus,&numberOfJobsInQueue, &numberOfServers,
    serverObjectIDList, clientConnIDList );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue for which the status is being obtained.
queueStatus	Passes a pointer to the space allocated for the queue status.
numberOfJobsInQueue	Passes a pointer to the space allocated for the number of jobs in the queue.
numberOfServers	Passes a pointer to the space allocated for the number of attached queue servers.
serverObjectIDList	Passes a pointer to an array allocated for queue server object IDs associated with the numberOfServers parameter.
clientConnIDList	Passes a pointer to an array allocated for clientConnIDs corresponding to the servers returned by the serverObjectIDList parameter.

Output

queueStatus	Receives the status of the specified queue. (See Appendix A, Queue Status Flags.)
numberOfJobsInQueue	Receives the number of jobs currently in the queue.

numberOfServers	Receives the number of attached queue servers.
serverObjectIDList	Receives an array of server IDs associated with the numberOfServers parameter.
clientConnIDList	Receives an array of clientConnIDs corresponding to the servers returned by the serverObjectIDList parameter.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0xFC	No Such Object
0x30	Invalid Connection ID
0xFF	Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function is a queue server function which reads the current status of the specified queue. The queueStatus parameter indicates the overall status of the queue. (See Appendix A, Queue Status Flags.)

The numberOfJobsInQueue parameter contains a count of the number of jobs currently in the queue, 0 to 250.

The numberOfServers parameter contains a count of the number of queue servers currently attached to service this queue, 0 to 25.

The serverObjectIDList and clientConnIDList parameters list queue servers currently servicing the queue by the queue server's objectID and the queue server's current workstation attachment (clientConnID).

Notes

Workstations making this call must be security equivalent to one of the objects listed in the queue's Q_USERS or Q_OPERATORS group properties.

See Also

NWAttachQueueServerToQueue
NWDetachQueueServerFromQueue
NWReadQueueServerCurrentStatus
NWSetQueueCurrentStatus
NWSetQueueServerCurrentStatus

NWReadQueueJobEntry

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function retrieves information about a specified queue job.

Synopsis

```
#include "nwapi.h"

int                     ccode;
uint16                  serverConnID;
uint32                  queueID;
uint16                  jobNumber;
NWQueueJobStruct_t      jobStruct;

ccode=NWReadQueueJobEntry( serverConnID, queueID, jobNumber,
    &jobStruct );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue associated with the queue job being read.
jobNumber	Passes the number of the job being read.
jobStruct	Passes a pointer to the structure (NWQueueJobStruct_t) allocated for queue job information.

Output

jobStruct	Receives the structure containing the queue job information. (See Appendix A, NWQueueJobStruct_t Structure.)

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0xD0	Queue Error
0xFC	No Such Object
0xD1	No Queue
0x30	Invalid Connection ID
0xD3	No Queue Rights
0xFF	Failure
0xD5	No Queue Job

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows an application to retrieve information about a job from a queue. The job's full 256-byte record is returned. See NWQueueJobStruct_t structure in Appendix A.

Notes

Workstations making this call must be security equivalent to one of the objects listed in the queue's Q_USER or Q_OPERATORS group properties.

See Also

NWChangeQueueJobEntry
NWChangeQueueJobPosition
NWCreateQueueFile
NWGetQueueJobList

NWReadQueueServerCurrentStatus

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	

This function reads the current status of a queue server.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint32    queueID;
uint32    queueServerID;
uint16    queueServerClientConnID;
void      serverStatusRecord[NWMAX_SERVER_STATUS_RECORD_
          LENGTH];

ccode=NWReadQueueServerCurrentStatus( serverConnID, queueID,
    queueServerID, queueServerClientConnID, serverStatusRecord );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue being affected.
queueServerID	Passes the bindery object ID of the queue server whose current status is being read.
queueServerClientConnID	Passes the connection number of the queue server being read.
serverStatusRecord	Passes a pointer to the buffer allocated for the status of the specified queue server.

Output

serverStatusRecord	Receives a the status of the specified queue server (64 bytes).

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job

0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows a station to read the current status of a queue server. The QMS maintains a 64-byte status record for each queue server attached to a queue.

The QMS does not interpret the contents of the status record. The record contains information important to the calling application. We recommend that the first 4 bytes of this record contain an estimated price for the given server to complete a "standard" job.

Notes

Workstations making this call must be security equivalent to one of the objects listed in the queue's Q_USER or Q_OPERATORS group properties.

See Also

NWSetQueueServerCurrentStatus

NWRemoveJobFromQueue

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function removes a job from a queue.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint32      queueID;
uint16      jobNumber;

ccode=NWRemoveJobFromQueue( serverConnID, queueID,
    jobNumber );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue where the job to be removed is located.
jobNumber	Passes the number of the job being removed.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows the workstation to remove a job from a queue. The jobNumber parameter contains the job number returned by the QMS when the job was created. The job number can also be obtained by using the NWGetQueueJobList function.

The specified job is removed from the queue, and the job file is closed and deleted. If the job is being serviced, the service is aborted. Further I/O requests made to the job's queue file return an ILLEGAL_FILE_HANDLE error.

Notes

Both the job's creator and an operator can call this function.

See Also

NWChangeQueueJobEntry
NWChangeQueueJobPosition
NWCreateQueueFile
NWGetQueueJobList
NWReadQueueJobEntry

NWRestoreQueueServerRights

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function restores a server's own identity after it has assumed its client's rights.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;

ccode=NWRestoreQueueServerRights( serverConnID );
```

Input

serverConnID	Passes the file server's connection ID.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows a queue server to restore its own identity after it has assumed its client's identity using the NWChangeToClientRights function. The queue server's login user identification and associated security equivalence list are restored to its original values.

This function does not change any of the path mappings (directory bases) held by the queue server. However, access rights to those directories are adjusted to reflect the rights the queue server has in those directories.

If the queue server has changed some of its path mappings as part of its efforts to service the queue job, the queue server must restore those directory bases.

Files opened using the client's rights before this function is called continue to be accessible with the client's rights. Files opened after this function is called are accessible only with rights of the queue server.

Notes

Only queue servers that have previously changed their identity, using the NWChangeToClientRights function, can call this function.

See Also

NWChangeToClientRights

NWServiceQueueJob

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	

This function allows a queue server to select a new job for servicing.

Synopsis

```
#include "nwapi.h"

int                     ccode;
uint16                  serverConnID;
uint32                  queueID;
uint16                  targetJobType;
NWQueueJobStruct_t      jobStruct;
NWFileHandle_ta         fileHandle;

ccode=NWServiceQueueJob( serverConnID, queueID, targetJobType,
    &jobStruct, fileHandle );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue whose jobs are being serviced.
targetJobType	Passes the type of the job to be serviced.
jobStruct	Passes a pointer to the job record of the next available job returned by the QMS. (See Appendix A, NWQueueJobStruct_t Structure.)
fileHandle	Passes a pointer to the file handle for the file associated with the job to be serviced.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows a queue server to select a new job for servicing.

Notes

The requesting client must have previously established itself as a queue server for the target queue.

See Also

NWAbortServicingQueueJob
NWAttachQueueServerToQueue
NWCreateQueueFile
NWFinishServicingQueueJob

NWSetQueueCurrentStatus

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function modifies a queue's status.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint32      queueID;
uint8       queueStatus;

ccode=NWSetQueueCurrentStatus( serverConnID, queueID,
    queueStatus );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue whose status is being updated.
queueStatus	Passes the control byte that determines the new status. (See Appendix A, Queue Status Flags.)

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0xD3	No Queue Rights
0xFC	No Such Object
0x30	Invalid Connection ID
0xFF	Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows the operator to control the addition of jobs and servers to the queue.

Notes

The client making this call must be logged in as one of the objects listed in the Q_OPERATORS property. The requesting client can become a queue operator by specifying its objectID when creating the queue (NWCreateQueue) or by adding its objectID with NWCreateProperty (see the Bindery Services chapter).

See Also

NWAttachQueueServerToQueue
NWCreateQueue
NWDetachQueueServerFromQueue
NWReadQueueCurrentStatus

NWSetQueueServerCurrentStatus

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function updates QMS's copy of a server's status record.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint32      queueID;
void        serverStatusRecord[NWMAX_SERVER_STATUS_RECORD_
                LENGTH];

ccode=NWSetQueueServerCurrentStatus( serverConnID, queueID,
    serverStatusRecord );
```

Input

serverConnID	Passes the file server connection ID.
queueID	Passes the bindery object ID of the queue to which the specified queue server is attached.
serverStatusRecord	Passes a pointer to the buffer containing the new status record of the queue server (64 bytes).

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0x99 (153) Directory Full
0xD0 (208) Queue Error
0xD1 (209) No Queue
0xD2 (210) No Queue Server
0xD3 (211) No Queue Rights
0xD4 (212) Queue Full
0xD5 (213) No Queue Job
0xD6 (214) No Job Right
0xD7 (215) Queue Servicing
0xD8 (216) Queue Not Active
0xD9 (217) Station Not Server
0xDA (218) Queue Halted
0xDB (219) Max. Queue Servers
0xFF (255) Failure

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The QMS does not interpret the contents of the status record. The record contains information important to the calling application only. We recommend that the first 4 bytes of this record contain an estimated price for the given server to complete a "standard" job.

Notes

Only workstations that have previously been attached to the queue as a queue server can make this call.

See Also

NWReadQueueServerCurrentStatus

End of Chapter

Chapter 7
Server Platform Service APIs

Function Calls

The Server Platform function calls allow developers to report file server information. Using these function calls, an application can also enable or disable login capabilities and determine file server disk usage.

This chapter discusses the following Server Platform Service APIs.

NWDisableServerPlatformLogin

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function disables the ability of clients to login to the specified server.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;

ccode=NWDisableServerPlatformLogin( serverConnID );
```

Input

serverConnID	Passes the file server connection ID.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC6	No Console Operator Rights
0x96	Server Out Of Memory
0xFB	Invalid Parameters
0xFF	No Response From Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

This call is useful after logging out all clients and then working with the server such as during backup procedures.

Notes

IMPORTANT: You must remember to enable server login before logging out of the server or no one, including the client that made this call will be able to access the server again.

See Also

NWEnableServerPlatformLogin

NWDownServerPlatform

NetWare 2.x	NetWare 3.x	NetWare for AVIiON Systems
✓	✓	✓

This function will down the specified server.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint8     forceFlag;

ccode=NWDownServerPlatform( serverConnID, forceFlag );
```

Input

serverConnID — Passes the file server connection ID of the server that will be shutdown

forceFlag — Passes one of the following flags:

0x00 - If any files are open, the call returns 0xFF and the server will not go down.

0xFF - Server shuts down after automatically closing all open files.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC6	No Console Operator Rights
0x96	Server Out Of Memory
0xFB	Invalid Parameters
0xFF	No Response From Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

This server will check for any open files and will not go down unless you set the force flag to 0xFF in which case all unfinished transaction will be backed out and all open files will be closed. The operating system will then be shut down and must be re-booted or brought up from the command line.

NWEnableServerPlatformLogin

NetWare 2.x	NetWare 3.x	NetWare for AVilON Systems
✓	✓	✓

This function enables clients to login to the specified server

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;

ccode=NWEnableServerPlatformLogin( serverConnID );
```

Input

serverConnID	Passes the file server connection ID.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC6	No Console Operator Rights
0x96	Server Out Of Memory
0xFB	Invalid Parameters
0xFF	No Response From Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

This function enables client login. If clients are already logged in when this function call is made there will be no interruption of service to those clients.

See Also

NWDisableServerPlatformLogin

NWGetDiskUtilization

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function call returns information about the total disk usage of a specified client.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint8     volNumber;
uint32    trusteeID;
uint16    dirCount;
uint16    fileCount;
uint16    blockCount;


ccode=NWGetDiskUtilization( serverConnID, volNumber, trusteeID,
    &dirCount, &fileCount, &blockCount );
```

Input

serverConnID	Passes the file server connection ID.
volNumber	Passes the number of the volume being checked.
trusteeID	Passes the client trustee ID (objectID of the trustee).
dirCount	Passes a pointer to the space allocated for the directory count.
fileCount	Passes a pointer to the space allocated for the file count.
blockCount	Passes a pointer to the space allocated for the cluster count.

Output

dirCount	Receives the number of directories owned by the trustee in the specified volume.
fileCount	Receives the number of files in the specified volume owned by the trustee.
blockCount	Receives the number of blocks used in the specified volume by the trustee.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC6	No Console Operator Rights
0x96	Server Out Of Memory
0xFB	Invalid Parameters
0xFF	No Response From Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

This function returns disk utilization information about a specified client (or trustee) based on their object ID number. NetWare blocks are usually 4Kb, but can be configured otherwise when the file server is initially configured. The NWGetVolInfoWithHandle (in File Services) returns exact information on the actual size of blocks on the given file server.

Notes

This call is not valid for NetWare for AViiON Systems servers. NetWare for AViiON Systems servers do not maintain this information on a per-client (objectID) basis.

See Also

NWGetVolInfoWithHandle

NWGetServerPlatformDateAndTime

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the network date and time maintained on the specified file server.

Synopsis

```
#include "nwapi.h"

int                                 ccode;
uint16                              serverConnID;
NWServerPlatformDateAndTime_t       dateAndTime;

ccode=NWGetServerPlatformDateAndTime( serverConnID, &dateAnd
Time );
```

Input

serverConnID	Passes the file server connection ID.
dateAndTime	Passes a pointer to the structure allocated for the network date and time. (See "Description," below, and Appendix A , NWServerPlatformDateAndTime_t Structure.)

Output

dateAndTime	Receives network date and time. (See "Description," below, and Appendix A, NWServerPlatformDateAndTime_t Structure.)

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC6	No Console Operator Rights
0x96	Server Out Of Memory
0xFB	Invalid Parameters
0xFF	No Response From Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

The NWServerPlatformDateAndTime_t structure is as follows:

```
typedef struct {
    uint8 year          (0 through 99; for example: 82=1982; )
    uint8 month                 (1 through 12)
    uint8 day           (1 through 31)
    uint8 hour                  (0 through 23)
    uint8 minute                (0 through 59)
    uint8 second                (0 through 59)
    uint8 dayOfWeek   (0 through 6, 0 = Sunday)
} NWServerPlatformDateAndTime_t;
```

Notes

Date and time are not automatically synchronized across an internetwork.

If the year is less than 82, the year is considered to be in the 21st century

See Also

NWSetServerPlatformDateAndTime

NWGetServerPlatformDescriptionStrings

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns descriptive information about a file server.

Synopsis

```
#include "nwapi.h"

int                     ccode;
uint16                  serverConnID;
NWDescriptionStrings_t  strings;

ccode=NWGetServerPlatformDescriptionStrings( serverConnID, &strings );
```

Input

serverConnID	Passes the file server connection ID.
strings	Passes a pointer to the structure allocated for the file server description stings. (See "Description" below and Appendix A, NWDescriptionStrings_t Structure.)

Output

strings	Receives the file server description strings. (See "Description" below and Appendix A, NWDescriptionStrings_t Structure.)

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC6	No Console Operator Rights
0x96	Server Out Of Memory
0xFB	Invalid Parameters
0xFF	No Response From Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

The NWDescriptionStrings_t structure is as follows:

```
typedef struct {
  char  companyName[ NWMAX_COMPANY_NAME_LENGTH ];
  char  revisionDescription[ NWMAX_DESCRIPTION_LENGTH ];
  char  revisionDate[ NWMAX_DATE_LENGTH ];
  char  copyrightNotice[ NWMAX_COPYRIGHT_NOTICE_LENGTH ];
} NWDescriptionStrings_t;
```

The companyName parameter receives the name of the company that is providing this version of NetWare. The revisionDescription parameter receives the NetWare version and revision description string. The revisionDate parameter receives the revision date in the form 02/15/1988. The copyrightNotice parameter passes a pointer to the string allocated for the copyright notice.

Notes

The requesting workstation must be attached to the file server. Console operator rights are not required to perform this function.

NWGetServerPlatformInformation

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓		✓

This function obtains information about the specified file server.

Synopsis

```
#include "nwapi.h"

int                    ccode;
uint16                 serverConnID;
NWServerPlatformInfo_t serverInfo;

ccode=NWGetServerPlatformInformation( serverConnID, &serverInfo );
```

Input

serverConnID	Passes the file server connection ID.
serverInfo	Passes a pointer to the structure allocated for the server information. (See "Description" below and Appendix A, NWServerPlatformInfo_t Structure.)

Output

serverInfo	Receives the server information. (See "Description" below and Appendix A, NWServerPlatformInfo_t Structure.)

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC6	No Console Operator Rights
0x96	Server Out Of Memory
0xFB	Invalid Parameters
0xFF	No Response From Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

This routine optionally returns several items of information about a file server based on a specified file server connection ID. If a certain item is not wanted, a NULL parameter can be substituted in place of the unwanted parameter. However, all parameter positions must be occupied.

The NWServerPlatformInfo_t structure is as follows:

```
typedef struct {
    uint16  majorVersion;
    uint16  minorVersion;
    uint16  revision;
    uint16  SFTLevel;
    uint16  TTSLevel;
    uint16  accountingVersion;
    uint16  VAPVersion;
    uint16  queueingVersion;
    uint16  printServerVersion;
    uint16  virtualConsoleVersion;
    uint16  securityRestrictionLevel;
    uint16  internetBridgeSupport;
    uint16  maxClientConnSupported;
    uint16  clientConnInUse;
    uint16  peakClientConnUsed;
    uint16  maxVolumes;
    char    serverName[ NWMAX_SERVER_NAME_LENGTH ];
} NWServerPlatformInfo_t;
```

See Also

NWGetServerPlatformDescriptionStrings

NWGetServerPlatformLoginStatus

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function gets the status of the server login—enabled or disabled.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint8     userLoginAllowed;

ccode=NWGetServerPlatformLoginStatus( serverConnID,
    &userLoginAllowed );
```

Input

serverConnID — Passes the file server connection ID.

userLoginAllowed — Passes a pointer to the space allocated for the status of user login.

Output

userLoginAllowed — Receives the status of user login:

0 = User Login Disabled
Non-zero = User Login Enabled

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC6	No Console Operator Rights
0x96	Server Out Of Memory
0xFB	Invalid Parameters
0xFF	No Response From Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

The userLoginAllowed parameter receives a non zero value if user login is enabled, and a 0 if user login is disabled. You must be a supervisor or supervisor equivalent to make this call.

See Also

NWEnableServerPlatformLogin
NWDisableServerPlatformLogin

NWGetServerPlatformName

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function gets the name of the server platform with the connection ID.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
char      serverPlatformName[NWMAX_SERVER_NAME_LENGTH];

ccode=NWGetServerPlatformName( serverConnID, serverPlatformName );
```

Input

serverConnID	Passes the file server connection ID.
serverPlatformName	Passes a pointer to the string allocated for the server name.

Output

serverPlatformName	Receives the server name.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFB	Invalid Parameters
0xF8	Not Attached To Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

This call does not actually call the server, but accesses the tables maintained by the Netware for AViiON Systems API Library which keeps track of the server information returned by the NWAttachToServerPlatform call.

See Also

NWAttachToServerPlatform

NWIsNetWare386

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function checks whether a connected file server is running NetWare 3.x.

Synopsis

```
#define "nwapi.h"

NWBoolean_ts    ccode;
uint16          serverConnID;

ccode=NWIsNetWare386( serverConnID );
```

Input

serverConnID	Passes the file server connection ID.

Output

None.

Return Values

1 Server is running NetWare 3.x
0 Server is not running NetWare 3.x

Notes

NetWare for AViiON Systems corresponds to NetWare 3.x.

NWSetServerPlatformDateAndTime

NetWare 2.x	NetWare 3.x	NetWare for AVilON Systems
✓	✓	

This function sets the network date and time on the specified file server.

Synopsis

```
#include "nwapi.h"

int                                 ccode;
uint16                              serverConnID;
NWServerPlatformDateAndTime_t       dateAndTime;

ccode=NWSetServerPlatformDateAndTime( serverConnID,
        &dateAndTime );
```

Input

serverConnID	Passes the file server connection ID.
dateAndTime	Passes a pointer to the structure allocated for the network date and time. (See "Description," below, and Appendix A, NWServerPlatformDateAndTime_t Structure.)

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xC6	No Console Operator Rights
0x96	Server Out Of Memory
0xFB	Invalid Parameters
0xFF	No Response From Server

Note: See Appendix B for a listing of possible NetWare errors.

Description

The NWServerPlatformDateAndTime_t structure is as follows:

```
typedef struct {
    uint8 year          (0 through 99; for example: 82=1982; )
    uint8 month         (1 through 12)
    uint8 day           (1 through 31)
    uint8 hour          (0 through 23)
    uint8 minute        (0 through 59)
    uint8 second        (0 through 59)
    uint8 dayOfWeek     (0 through 6, 0 = Sunday)
} NWServerPlatformDateAndTime_t;
```

Notes

Date and time are not automatically synchronized across an internetwork.

If the year is less than 82, the year is considered to be in the 21st century

This call is not valid for NetWare for AViiON Systems servers. If the system date and time need to be changed, the host administrator should be contacted.

See Also

NWGetServerPlatformDateAndTime

End of Chapter

Chapter 8
Synchronization Service APIs

Function Calls

This chapter describes of the following Synchronization Service APIs. They can be categorized as follows: [L] logical record locks, [P] physical record locks, [F] file locks, [S] semaphores.

Introduction to Synchronization Services

Synchronization services calls enable applications to coordinate access to network files and other resources. These services are divided into two categories: Locking and Semaphores.

Locking and Semaphores

NetWare provides calls that allow applications to lock files, physical records, or logical records. The client must maintain tables which keep track of file and record locks. There are no APIs which can return this information.

Before locking a file/record, an client should also record the following information about the file/record in a File Log table residing on the file server:

- Name (for files)
- Location and Size (for records)

There are two kinds of locks available to application developers - physical and logical. Unlike a physical record lock, a logical record lock does not actually lock bytes. Instead, a logical record lock acts somewhat like a semaphore. Applications cooperatively define a logical record name that represents a group of files, records, structures, etc. When an application locks a logical record, it only locks the logical record name, and not the group of files, records, or structures the name represents. Any uncooperative application can ignore a lock on the logical record name and directly lock the physical files or records. Therefore, applications using logical record locks should not use other locking techniques simultaneously on the same data.

NetWare also provides calls that enable applications to create, open, examine, and close semaphores. Applications can also use semaphore calls to increment and decrement the value associated with a semaphore.

NWClearFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks the specified file and removes it from the File Log table.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;

ccode=NWClearFile( &path );
```

Input

path	Passes a pointer to the name of the file whose file lock is being cleared.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

If the file is open, this function closes it and invalidates the file handle. The file can then be accessed by other clients. If the calling client had the file open multiple times, it is closed as many times and all associated file handles are invalidated.

Notes

The fileName parameter can specify either a file's complete path name or a path relative to the default directory mapping.

See Also

NWClearFileSet

NWClearFileSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks all the files logged for a specified client.

Synopsis

```
#include "nwapi.h"

int     ccode;

ccode=NWClearFileSet( );
```

Input

None.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

All open files in the client's File Log table are closed and the file handles are cleared. This includes files logged across any servers that have been logged into the set.

Notes

This function is ignored if the requesting client does not have logged files.

See Also

NWClearFile
NWLockFileSet
NWLogFile
NWReleaseFile
NWReleaseFileSet

NWClearLogicalRecord

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks a logical record and removes it from the logical record log table.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
char        logicalRecordName[NWMAX_LOGICAL_RECORD_NAME_
                LENGTH];

ccode=NWClearLogicalRecord( serverConnID, logicalRecordName );
```

Input

serverConnID	Passes the file server connection ID.
logicalRecordName	Passes a pointer to the name of the logical record being cleared (128 characters).

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

This function clears on record residing in the Logical Record Log table associated exclusively with the requesting client.

Applications define logical record names. A logical record name represents a group of files, physical records, structures, etc. The NWLogLogicalRecord or NWLockLogicalRecordSet functions locks one or more logical record names, not the actual files, physical records, or structures associated with each logical record name. Any uncooperative application can ignore a lock on the logical record name and directly lock physical files or records.

Therefore, applications using logical record locks must not use other locking techniques simultaneously.

See Also

NWClearLogicalRecordSet
NWLockLogicalRecordSet
NWLogLogicalRecord
NWReleaseLogicalRecord
NWReleaseLogicalRecordSet

NWClearLogicalRecordSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks and removes all logical records in the Logical Record Log table (on any of the servers you are attached to).

Synopsis

```
#include "nwapi.h"

int     ccode;

ccode=NWClearLogicalRecordSet( );
```

Input

None.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

This function unlocks all of the logical records that are logged in the Logical Record Log table and removes them from the table. If the requesting process does not have logged logical records, this function is ignored.

See Also

NWClearLogicalRecord
NWLockLogicalRecordSet
NWLogLogicalRecord
NWReleaseLogicalRecord
NWReleaseLogicalRecordSet

NWClearPhysicalRecord

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks the specified physical record and removes it from the Log table.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
NWFileHandle_ta fileHandle;
uint32          recordStartOffset;
uint32          recordLength;

ccode=NWClearPhysicalRecord( serverConnID, fileHandle, recordStartOffset,
    recordLength);
```

Input

serverConnID	Passes the file server connection ID.
fileHandle	Passes the file handle associated with the file containing the physical record being cleared.
recordStartOffset	Passes the offset, within the file, where the locked record begins.
recordLength	Passes the length, in bytes, of the locked record.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

The Record Log table resides on the file server and is associated exclusively with the requesting task on the workstation.

The application locates the physical record within the specified file by passing the offset in the recordStartOffset parameter and the length in the recordLength parameter.

Notes

This function is ignored if the requesting workstation does not have logged physical records.

See Also

NWClearPhysicalRecordSet
NWLockPhysicalRecordSet
NWLogPhysicalRecord
NWReleasePhysicalRecord
NWReleasePhysicalRecordSet

NWClearPhysicalRecordSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks and removes all physical records from the Physical Record Log table.

Synopsis

```
#include "nwapi.h"

int     ccode;

ccode=NWClearPhysicalRecordSet( );
```

Input

None.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

This function clears an entire set of physical records associated with the requesting client.

Notes

This function is ignored if the requesting workstation does not have logged physical records.

See Also

NWClearPhysicalRecord
NWLockPhysicalRecordSet
NWLogPhysicalRecord
NWReleasePhysicalRecord
NWReleasePhysicalRecordSet

NWCloseSemaphore

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function closes a semaphore.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
uint32          NetWareSemaphoreHandle;

ccode=NWCloseSemaphore( serverConnID, NetWareSemaphoreHandle );
```

Input

serverConnID	Passes the file server connection ID.
NetWareSemaphoreHandle	Passes the semaphore handle obtained when the semaphore was opened (NWOpenSemaphore).

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

This function decrements the open count of the semaphore, indicating that one less process is holding the semaphore open. If the requesting process is the last process to have this semaphore open, the semaphore is deleted.

See Also

NWExamineSemaphore
NWOpenSemaphore
NWSignalSemaphore
NWWaitOnSemaphore

NWExamineSemaphore

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the current value and open count for the specified semaphore.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
uint32          semaphoreHandle;
int             semaphoreValue;
uint16          semaphoreOpenCount;

ccode=NWExamineSemaphore( serverConnID, semaphoreHandle,
        &semaphoreValue, &semaphoreOpenCount );
```

Input

serverConnID	Passes the file server connection ID.
semaphoreHandle	Passes the semaphore handle obtained when the semaphore was opened (NWOpenSemaphore).
semaphoreValue	Passes a pointer to the space allocated for the current semaphore value (optional).
semaphoreOpenCount	Passes a pointer to the space allocated for the number of stations that currently have this semaphore open.

Output

semaphoreValue	Receives the current semaphore value (optional).
semaphoreOpenCount	Receives the number of stations that currently have this semaphore open.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

The semaphore value is decremented for each NWWaitOnSemaphore function call and incremented for each NWSignalSemaphore function call.

A positive semaphore value indicates that the application can access the associated network resource. A negative value indicates the number of processes waiting to use the semaphore. If the semaphore value is negative, the application must either enter a waiting queue by calling the function NWWaitOnSemaphore or temporarily abandon its attempt to access the network resource.

The semaphoreOpenCount parameter indicates the number of processes holding the semaphore open. A call to NWOpenSemaphore increments this value. A call to NWCloseSemaphore decrements this value.

See Also

NWCloseSemaphore
NWOpenSemaphore
NWSignalSemaphore
NWWaitOnSemaphore

NWLockFileSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function locks all files that are logged in the File Log table.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          timeOutLimit;

ccode=NWLockFileSet( timeOutLimit );
```

Input

timeOutLimit — Passes the length of time the file server waits.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

The timeOutLimit indicates how long the file server will attempt to lock the set. This limit is specified in units of 1/18 of a second (0 = no wait).

See Also

NWClearFile
NWClearFileSet
NWLogFile
NWReleaseFile
NWReleaseFileSet

NWLockLogicalRecordSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	

This function locks all logical records logged in the Logical Record Log table.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint8     lockFlags;
uint16    timeOutLimit;

ccode=NWLockLogicalRecordSet( lockFlags, timeOutLimit );
```

Input

lockFlags	Passes one of the following lock flags: NWLS_EXCLUSIVE NWLS_SHAREABLE
timeOutLimit	Passes the length of time the file server attempts to lock the record set before timing out.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

Applications define logical record names. A logical record name represents a group of files, physical records, or data structures. NWLogLogicalRecord or NWLockLogicalRecordSet function affects one or more logical record names, not the actual files, physical records, or data structures associated with each logical record name. Any uncooperative application can ignore a lock on the logical record name and directly lock physical files or records. Therefore, applications using logical record locks must not simultaneously use other locking techniques.

The timeOutLimit parameter indicates how long the file server will attempt to lock the logical record set. The timeOutLimit parameter is specified in units of 1/18 of a second (0 = no wait).

See Also

NWClearLogicalRecord
NWClearLogicalRecordSet
NWLogLogicalRecord
NWReleaseLogicalRecord
NWReleaseLogicalRecordSet

NWLockPhysicalRecordSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function locks all records logged in the Physical Record Log table.

Synopsis

```
#include "nwapi.h"

int     ccode;
uint8   lockFlags;
uint16  timeOutLimit;

ccode=NWLockPhysicalRecordSet( lockFlags, timeOutLimit );
```

Input

lockFlags — Passes one of the following lock flags:

NWLS_EXCLUSIVE
NWLS_SHAREABLE

timeOutLimit — Passes the length of time the file server attempts to lock the physical record set before timing out.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

The timeOutLimit is specified in units of 1/18 of a second (0 = no wait).

Notes

This function cannot lock a record that is already locked exclusively by another application. If one or more records, identified in the log table, are already exclusively locked by another application, the attempt to lock the set fails.

See Also

NWClearPhysicalRecord
NWClearPhysicalRecordSet
NWLogPhysicalRecord
NWReleasePhysicalRecord
NWReleasePhysicalRecordSet

NWLogFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function logs the file in the File Log table. Use this function when you need to lock a set of files. If the lock flag is set, this function also locks the file.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;
uint8           logFlags;
uint16          timeOutLimit;

ccode=NWLogFile( &path, logFlags, timeOutLimit );
```

Input

path	Passes a pointer to the file name to be locked (must be the full path name for the file - up to 255 characters).
logFlags	Passes one of the following log flags. NWFL_LOG_ONLY NWFL_LOG_AND_LOCK
timeOutLimit	Passes the length of time the file server attempts to log the specified file before timing out.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. one of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

When the logFlags parameter is set to NWFL_LOG_AND_LOCK, the server will attempt to lock a file for the length of time specified by the timeOutLimit parameter. The timeOutLimit parameter is specified in units of 1/18 of a second (0 = no wait).

A log table contains data locking information used by a file server. The file server tracks this information for each workstation and process. Whenever a file, logical record, or physical record is logged, information identifying the data being logged is placed in the log table. Normally, a set of files or records is logged and then locked as a set. However, a single file or record can also be locked when it is placed in the log table.

When using log tables, a task first logs all of the files or records that are needed to complete a transaction. The task then attempts to lock the logged set of files or records. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

See Also

NWClearFile
NWClearFileSet
NWLockFileSet
NWReleaseFile

NWLogLogicalRecord

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	

This function logs a logical record.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
char            logicalRecordName[NWMAX_LOGICAL_RECORD_NAME_
                    LENGTH];
uint8           lockFlags;
uint16          timeOutLimit;

ccode=NWLogLogicalRecord( serverConnID, logicalRecordName, lockFlags,
        timeOutLimit );
```

Input

serverConnID	Passes the file server connection ID.
logicalRecordName	Passes a pointer to the name of the logical record being logged (128 characters).
lockFlags	Passes one of the following lock flags. NWPL_LOG_ONLY NWPL_LOG_AND_LOCK_EXCLUSIVE NWPL_LOG_AND_LOCK_SHAREABLE
timeOutLimit	Passes the length of time the file server attempts to lock the record before timing out.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

When the lockFlags parameter is set to option one or three, the file server will attempt to lock the logical record for the length of time specified by the timeOutLimit parameter. The timeOutLimit parameter is specified in units of 1/18 of a second.

A log table contains data locking information used by a file server. The file server tracks this information for each workstation and workstation task. Whenever a file, logical record, or physical record is logged, information identifying the data being logged is placed in the log table.

Normally, a set of files or records is logged and then locked as a set. However, a single file or record can also be locked when it is placed in the log table. The release functions are used to unlock a lock (or set of locks). The clear functions are used to unlock and remove a lock (or set of locks) from the log table.

When using log tables, a task first logs all files or records to complete a transaction. The task attempts to lock the logged set of files or records. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

See Also

NWClearLogicalRecord
NWClearLogicalRecordSet
NWLockLogicalRecordSet
NWReleaseLogicalRecord
NWReleaseLogicalRecordSet

NWLogPhysicalRecord

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function logs a physical record in the Physical Record Log table.

Synopsis

```
#include "nwapi.h"

int                 ccode;
uint16              serverConnID;
NWFileHandle_ta     fileHandle;
uint32              recordStartOffset;
uint32              recordLength;
uint8               lockFlags;
uint16              timeOutLimit;

ccode=NWLogPhysicalRecord( serverConnID, fileHandle, recordStartOffset,
    recordLength, lockFlags, timeOutLimit );
```

Input

serverConnID	Passes the file server connection ID.
fileHandle	Passes the file handle of the file whose record is being logged.
recordStartOffset	Passes the offset into the file where the record being logged begins.
recordLength	Passes the length, in bytes, of the record to be logged.
lockFlags	Passes one of the following lock flags: NWPL_LOG_ONLY NWPL_LOG_AND_LOCK_EXCLUSIVE NWPL_LOG_AND_LOCK_SHAREABLE
timeOutLimit	Passes the length of time the file server attempts to log a record before timing out.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

The file server attempts to log the record for the length of time specified by the timeOutLimit parameter before returning a time out error. The timeoutLimit parameter is specified in units of 1/18 of a second (0 = no wait).

A log table contains data locking information used by a file server. The file server tracks this information for each workstation and process. Whenever a file, logical record, or physical record is logged, information identifying the data being logged is placed in the log table. Normally, a set of files or records is logged and then locked as a set. However, a single file or record can also be locked when it is placed in the log table.

The release functions are used to unlock a lock or set of locks.

The clear functions are used to unlock and remove a lock or set of locks from the log table.

Notes

When using log tables, a task first logs all files or records to complete a transaction. The task then attempts to lock the logged set of files or records. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

See Also

NWClearLogicalRecord
NWClearLogicalRecordSet
NWLockLogicalRecordSet
NWReleaseLogicalRecord
NWReleaseLogicalRecordSet

NWOpenSemaphore

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function opens a semaphore.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
char        semaphoreName[NWMAX_SEMAPHORE_NAME];
int         initialSemaphoreValue;
uint32      NetWareSemaphoreHandle;
uint16      semaphoreOpenCount;

ccode=NWOpenSemaphore( serverConnID, semaphoreName,
    initialSemaphoreValue, &NetWareSemaphoreHandle,
    &semaphoreOpenCount );
```

Input

serverConnID	Passes the file server connection ID.
semaphoreName	Passes a pointer to the name of the semaphore to be opened.
initialSemaphoreValue	Passes the initial value of the semaphore being opened (must be greater than 1).
NetWareSemaphoreHandle	Passes a pointer to the space allocated for the NetWare semaphore handle.
semaphoreOpenCount	Passes a pointer to the space allocated for the number of stations that have this semaphore opened (optional).

Output

NetWareSemaphoreHandle	Receives the NetWare semaphore handle.
semaphoreOpenCount	Receives the number of stations that have this semaphore opened (optional).

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

This function creates and initializes the semaphore to the value indicated by the initialSemaphoreValue parameter.

The semaphoreOpenCount parameter indicates the number of processes currently using the semaphore. Semaphores can have two purposes under NetWare.

1. Semaphores can limit the number of users to a particular resource. To limit the number of users, use the NWOpenSemaphore and NWCloseSemaphore function calls only.

   The NWOpenSemaphore call returns the number of processes that currently have the semaphore open. An application can check the value against a programmer-defined limit and take appropriate action.

2. Semaphores can restrict access to a particular resource. If access is restricted, only serial access is allowed. To request access, a resource must open the semaphore associated with the resource and, through that semaphore, request permission to access the resource. If the resource is unavailable, the calling process is placed in a wait queue.

   Restricting access to a resource is not limited to allowing only one process to have access. For example, if the resource being restricted is a modem pool with four modems, the semaphore could allow access to four users but could restrict access to subsequent requesters by setting the initialSemaphoreValue parameter to four.

See Also

NWCloseSemaphore
NWExamineSemaphore
NWSignalSemaphore
NWWaitOnSemaphore

NWReleaseFile

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks the specified file but does not remove it from the File Log table.

Synopsis

```
#include "nwapi.h"

int             ccode;
NWPath_t        path;

ccode=NWReleaseFile( &path );
```

Input

path — Passes a pointer to the NWPath_t structure containing the directory handle, the server connection ID, and a pointer to the path name. (See Appendix A, NWPath_t Structure.)

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

See Also

NWClearFile
NWClearFileSet
NWLogFile
NWReleaseFileSet

NWReleaseFileSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks all files logged in the File Log table.

Synopsis

```
#include "nwapi.h"

int     ccode;

ccode=NWReleaseFileSet( );
```

Input

None.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

The NWReleaseFileSet function does not remove files from the log table. This function is ignored if the requesting workstation does not have locked files.

See Also

NWClearFile
NWClearFileSet
NWLogFile
NWReleaseFile

NWReleaseLogicalRecord

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks a logical record but does not remove it from the Logical Record Log table.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
char        logicalRecordName[NWMAX_LOGICAL_RECORD_NAME];

ccode=NWReleaseLogicalRecord( serverConnID, logicalRecordName );
```

Input

serverConnID	Passes the file server connection ID.
logicalRecordName	Passes a pointer to the name of the logical record being released (128 characters).

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

A log table contains data locking information used by a file server. The file server tracks this information for each workstation and workstation task. Whenever a file, logical record, or physical record is logged, information identifying the data being logged is placed in the log table. Normally, a set of files or records is logged and then locked as a set. However, a single file or record can also be locked when it is placed in the log table.

See Also

NWClearLogicalRecord

NWReleaseLogicalRecordSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks all the logical records in the Logical Record Log table.

Synopsis

```
#include "nwapi.h"

int     ccode;

ccode=NWReleaseLogicalRecordSet( );
```

Input

None.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

The NWReleaseLogicalRecordSet function does not remove logical records from the log table. This function is ignored if the requesting workstation or process does not have locked logical records.

See Also

NWClearLogicalRecord
NWClearLogicalRecordSet
NWLockLogicalRecordSet
NWLogLogicalRecord
NWReleaseLogicalRecord

NWReleasePhysicalRecord

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks the specified physical record but does not remove it from the log table.

Synopsis

```
#include "nwapi.h"

int                 ccode;
uint16              serverConnID;
NWFileHandle_ta     fileHandle;
uint32              recordStartOffset;
uint32              recordLength;

ccode=NWReleasePhysicalRecord( serverConnID, fileHandle,
    recordStartOffset, recordLength );
```

Input

serverConnID	Passes the file server connection ID.
fileHandle	Passes the file handle associated with the file containing the specified record.
recordStartOffset	Passes the offset, within the file, where the physical record begins.
recordLength	Passes the length, in bytes, of the record being released.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

See Also

NWClearPhysicalRecord
NWClearPhysicalRecordSet
NWLockPhysicalRecordSet
NWLogPhysicalRecord
NWReleasePhysicalRecordSet

NWReleasePhysicalRecordSet

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function unlocks all records logged in the Physical Record Log table but leaves them logged in the table.

Synopsis

```
#include "nwapi.h"

int          ccode;

ccode=NWReleasePhysicalRecordSet( );
```

Input

None.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Notes

This function is ignored if the workstation does not have locked physical records.

See Also

NWClearPhysicalRecord
NWClearPhysicalRecordSet
NWLockPhysicalRecordSet
NWLogPhysicalRecord
NWReleasePhysicalRecord

NWSignalSemaphore

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function signals a semaphore that the station or process is finished.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
uint32          semaphoreHandle;

ccode=NWSignalSemaphore( serverConnID, semaphoreHandle );
```

Input

serverConnID	Passes the file server connection ID.
NetWareSemaphoreHandle	Passes the semaphore handle pointing to the semaphore to be signaled.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

An application must call this function when it finishes accessing the network resource associated with the semaphore. If processes are waiting to use the semaphore, the first process in the queue is released (signaled).The NetWareSemaphoreHandle is a uint32 pointer to a semaphore. An application obtains this handle with a call to the NWOpenSemaphore function.

See Also

NWCloseSemaphore
NWExamineSemaphore
NWOpenSemaphore
NWWaitOnSemaphore

NWWaitOnSemaphore

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function decrements a semaphore value.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
uint32          semaphoreHandle;
uint16          timeOutLimit;

ccode=NWWaitOnSemaphore( serverConnID, semaphoreHandle,
    timeOutLimit );
```

Input

serverConnID	Passes the file server connection ID
NetWareSemaphoreHandle	Passes the semaphore handle returned from the NWOpenSemaphore function call.
timeOutLimit	Passes the length of time the application will be wait.

Output

None.

Return Values

0 Successful.
-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFD	Lock Collision
0xFE	Timeout Error
0xFF	Lock Error
0xFF	Unlock Error

Note: See Appendix B for a listing of possible NetWare errors.

Description

If the semaphore value is greater than or equal to zero, the application can access the associated resource. If the value is less than zero, the function queues the application for the time interval specified in the timeoutLimit parameter. If the semaphore value is greater than or equal to zero when the timeout value expires, the application can access the resource. If the semaphore value is still negative, the function removes the application from the queue and re-increments the semaphore value. A semaphore handle is a uint32 pointer to a semaphore and can be obtained by calling NWOpenSemaphore.

The timeout limit indicates how long the file server should wait if the semaphore value is negative. The timeout limit is specified in units of 1/18 of a second (0 = no wait).

See Also

NWCloseSemaphore
NWExamineSemaphore
NWOpenSemaphore
NWSignalSemaphore

End of Chapter

Chapter 9
Transaction Tracking Service APIs

Function Calls

This chapter discusses the Transaction Tracking System APIs that are listed below.

Note: Your version of NetWare for AViiON Systems may not support Transaction Tracking Services. Refer to the release notice accompanying your shipment for specific restrictions.

Introduction to Transaction Tracking

NetWare file servers that include Transaction Tracking System (TTS) can track transactions and ensure file integrity by backing out (or erasing) interrupted or partially completed transactions. TTS only affects to transactional files. A file becomes transactional when the file's Transactional extended file attribute is set.

For example, a banking database application frequently performs a transaction that includes the following three writes to database files:

- A debit to one account
- A credit to another account
- A note to a log

The application must complete all three of these writes to maintain database integrity. Transaction tracking is implemented in two ways, implicit and explicit.

- Implicit Transaction Tracking. Implicit Transaction Tracking requires no coding on the part of an application developer. If TTS is installed and enabled on a file server, TTS tracks all transactions to all transactional files (including transactions made by NetWare to bindery files).
- Explicit Transaction Tracking. Explicit Transaction Tracking has two calls: NWTTSBeginTransaction and NWTTSEndTransaction. Explicit Transaction Tracking requires applications to make TTS calls, and allows applications to neatly bracket file update sequences with locking and TTS calls. An application would most likely use logical or physical record locks with TTS calls (see the "Synchronization" section).

The following steps describe how TTS tracks each write within a transaction.

1. An application writes new data to a file on a file server.
2. The file server stores the new data in cache memory. The target file on the file server hard disk remains unchanged.
3. The file server scans the target file on the file server hard disk, finds the data to be changed (old data), and copies the old data to cache memory. The file server also records the name and directory path of the target file and the location and length of the old data (record) within the file. The target file on the file server hard disk still remains unchanged.
4. The file server writes the old data in cache memory to a transaction work file on the file server hard disk. The transaction work file resides at the root level of volume SYS on the file server. The file is flagged System and Hidden. The target file on the file server hard disk still remains unchanged.
5. The file server writes the new data in cache memory to the target file on the file server hard disk. The target file is now changed.

The file server repeats these steps for each write within a transaction. The transaction work file grows to accommodate the old data for each write. If the transaction is interrupted, the file server writes the contents of the transaction work file to the target file, thereby restoring the file to its pretransaction condition. In effect, the file server backs out the transaction.

A file server can monitor from 100 to 10,000 transactions at a time. (The maximum value can be configured with SET for NetWare 3.x.) A file server can track only one transaction at a time for each session. If a session sends several transactions to a file server rapidly, the file server queues the transactions and services them one at a time.

NWTTSAbortTransaction

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function aborts all transactions, explicit and implicit, on a file that has been flagged transactional.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;

ccode=NWTTSAbortTransaction( serverConnID );
```

Input

serverConnID — Passes the file server connection ID.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: A return value in NWErrno of 0xFE indicates that more than the threshold number of logical or physical records are still locked by the application. However, the transaction is still finished and any locks being held are released. In this case, the file server automatically starts a new implicit transaction.

See Appendix B for a complete listing of possible NetWare errors.

Description

This function releases the following record locks:

- Physical record locks generated by the file server when an application tried to write an unlocked record
- Physical or logical locks that have not been released because of a file write

When this function is complete, all transactions will have been successfully backed out.

Notes

If a transaction is aborted, all writes made since the beginning of a transaction are cancelled, and all files are returned to the state they were in before the transaction began.

Files can be flagged transactional with NWCreateFile and NWSetFileAttributes.

See Also

NWCreateFile
NWSetFileAttributes
NWTTSBeginTransaction
NWTTSEndTransaction

NWTTSBeginTransaction

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function begins an explicit transaction on a file that has been flagged transactional.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;

ccode=NWTTSBeginTransaction( serverConnID );
```

Input

serverConnID — Passes the file server connection ID.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function tracks all transactional files that are currently open, and all those that are opened during the transaction.

When data is written to a transactional file during a transaction, the file server automatically generates a physical record lock for the region being written. If a lock already exists, no additional lock is generated. This automatic locking can be disabled using the NWTTSSetControlFlags function.

Notes

Any closing and unlocking of transactional files is delayed until an NWTTSEndTransaction or NWTTSAbortTransaction is executed. Logical and physical records are not unlocked until the end of the transaction if file writes are performed while the lock is in force.
Use NWCreateFile or NWSetFileAttributes to flag a file transactional.

See Also

NWCreateFile
NWSetFileAttributes
NWTTSAbortTransaction
NWTTSSetControlFlags
NWTTSEndTransaction

NWTTSDisableTransactionTracking

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function disables transaction tracking services on the specified file server.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;

ccode=NWTTSDisableTransactionTracking( serverConnID );
```

Input

serverConnID — Passes the file server connection ID.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active
0xC6	No Console Privileges

Note: A return value in NWErrno of 0xFE indicates that more than the threshold number of logical or physical records are still locked by the application. However, the transaction is still finished and any locks being held are released. In this case, the file server automatically starts a new implicit transaction.

See Appendix B for a complete listing of possible NetWare errors.

Description

This function call should be used after transaction services are no longer being used.

The application making this call must be supervisor or have equivalent rights.

See Also

NWTTSEnableTransactionTracking
NWTTSBeginTransaction
NWTTSIsTransactionWritten

NWTTSEnableTransactionTracking

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function enables transaction tracking services on the specified file server.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;

ccode=NWTTSEnableTransactionTracking( serverConnID );
```

Input

serverConnID — Passes the file server connection ID.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active
0xC6	No Console Privileges

Note: A return value in NWErrno of 0xFE indicates that more than the threshold number of logical or physical records are still locked by the application. However, the transaction is still finished and any locks being held are released. In this case, the file server automatically starts a new implicit transaction.

See Appendix B for a complete listing of possible NetWare errors.

Description

This function call will enable TTS if the server has it available. However, the version of NetWare that the file server is running determines the information the call returns.

For NetWare 3.x, this call enables TTS. Use NWTTSIsAvailable to check whether TTS has been disabled.

For NetWare 2.x, this call enables TTS if TTS has been installed on the file server. If TTS has not been installed, the call will not fail. On file servers running NetWare 2.x, you should always use NWTTSIsAvailable before making this call.

Notes

The application making this call must be supervisor or have equivalent rights.

See Also

NWTTSDisableTransactionTracking
NWTTSBeginTransaction
NWTTSIsAvailable
NWTTSIsTransactionWritten

NWTTSEndTransaction

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function ends an explicit transaction on a file that has been flagged transactional. The function also returns a transaction reference number.

Synopsis

```
#include "nwapi.h"

int             ccode;
uint16          serverConnID;
uint32          transactionReferenceNumber;


ccode=NWTTSEndTransaction( serverConnID,
&transactionReferenceNumber );
```

Input

serverConnID	Passes the file server connection ID.
transactionReferenceNumber	Passes a pointer to the space allocated for the transaction reference number for the transaction being ended.

Output

transactionReferenceNumber	Receives the transaction reference number for the transaction being ended.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno.

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: A return value in NWErrno of 0xFE indicates that more than the threshold number of logical or physical records are still locked by the application. However, the transaction is still finished and any locks being held are released. In this case, the file server automatically starts a new implicit transaction.

See Appendix B for a complete listing of possible NetWare errors.

Description

The transaction is not necessarily written to disk when the reference number is returned, however a client must use the NWTTSIsTransactionWritten function to verify that a transaction has been written to disk. If the file server fails before all updates contained within the transaction have been written to disk, the transaction will be backed out when the file server is rebooted.

If transaction tracking is disabled, the reference number can still be used to determine when the transaction has been completely written to disk.

Notes

This function releases all physical record locks generated by the file server when a write is made to an unlocked record. In addition, physical or logical locks that have not been released because of a file write are unlocked at this time.

Files can be flagged transactional with NWCreateFile and with NWSetFileAttributes.

See Also

NWCreateFile
NWSetFileAttributes
NWTTSAbortTransaction
NWTTSBeginTransaction
NWTTSIsTransactionWritten

NWTTSGetConnectionThresholds

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the number of logical and physical record locks allowed for implicit transactions.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint8       logicalRecordLockThreshold;
uint8       physicalRecordLockThreshold;

ccode=NWTTSGetConnectionThresholds( serverConnID,
        &logicalRecordLockThreshold, &physicalRecordLockThreshold );
```

Input

serverConnID	Passes the file server connection ID.
logicalRecordLockThreshold	Passes a pointer to the space allocated for the number of logical record locks allowed before implicit transactions begin (0 to 255).
physicalRecordLockThreshold	Passes a pointer to the space allocated for the number of physical record locks allowed before implicit transactions begin (0 to 255).

Output

logicalRecordLockThreshold	Receives the number of logical record locks allowed before implicit transactions begin (0 to 255).
physicalRecordLockThreshold	Receives the number of physical record locks allowed before implicit transactions begin (0 to 255).

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function allows an application to get the number of logical and physical record locks allowed before implicit transactions begin.

The NWTTSSetConnectionThresholds function and this function are useful for applications that change the implicit application thresholds and later want to restore them. For example, NWTTSGetConnectionThresholds can get the number of logical and physical locks, and NWTTSSetConnectionThresholds can do one of the following:

- Turn off implicit transactions. (Applications that use only explicit transactions, but sometimes generate unnecessary implicit transactions, need to turn off all implicit transactions.)

- Set implicit thresholds for applications that always keep one or more records locked.

Notes

The default threshold for logical and physical locks is 0. A threshold of 255 means implicit transactions for that lock type have been completed.

See Also

NWTTSSetConnectionThresholds

NWTTSGetControlFlags

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
		✓

This function returns the control flags byte for files flagged as transactional.

Synopsis

```
#include "nwapi.h"

int       ccode;
uint16    serverConnID;
uint8     TTSControlFlags;

ccode=NWTTSGetControlFlags( serverConnID, &TTSControlFlags );
```

Input

serverConnID	Passes the file server connection ID.
TTSControlFlags	Passes a pointer to the space allocated for the Transaction Tracking Control flags. (See Description below.)

Output

TTSControlFlags	Receives a Transaction Tracking Control flag. (See Description below.)

Return Values

0	Successful.
-1	Unsuccessful. One of the following error codes is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

Transaction tracking control flags are only valid for files flagged as TTS (transactional). These control flags are defined as follows:

0x00	Automatic record locking is disabled
0x01	Automatic record locking is enabled

See Also

NWTTSSetControlFlags

NWTTSGetProcessThresholds

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function returns the number of explicit physical and logical record locks that can be done before implicit locking begins.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint8       logicalRecordLockThreshold;
uint8       physicalRecordLockThreshold;

ccode=NWTTSGetProcessThresholds( serverConnID,
        &logicalRecordLockThreshold, &physicalRecordLockThreshold );
```

Input

serverConnID	Passes the file server connection ID.
logicalRecordLockThreshold	Passes a pointer to space allocated for the number of explicit logical record locks allowed before implicit transactions begin.
physicalRecordLockThreshold	Passes a pointer to the space allocated for the number of explicit physical record locks allowed before implicit transactions begin.

Output

logicalRecordLockThreshold	Receives the number of explicit logical record locks allowed before implicit transactions begin.
physicalRecordLockThreshold	Receives the number of explicit physical record locks allowed before implicit transactions begin .

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function and the NWTTSSetProcessThresholds function are useful for applications that change the implicit process thresholds and later want to restore them. For example, NWTTSGetProcessThresholds can query an application for the number of logical and physical record locks allowed before an implicit transaction begins. NWTTSSetProcessThresholds can then do one of the following:

- Turn off implicit transactions
- Set implicit thresholds for applications that always keep one or more records locked

The default threshold for logical and physical locks is 0. A threshold of 255 means there will be no implicit transactions allowed for that lock type.

The thresholds returned by this function are valid for the requesting application only. When the application terminates, the connection thresholds are restored.

Notes

Applications that intend to use only explicit transactions, but sometimes generate unnecessary implicit transactions, need to turn off all implicit transactions.

See Also

NWTTSSetProcessThresholds

NWTTSIsAvailable

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function verifies that the file server supports transaction tracking.

Synopsis

```
#include "nwapi.h"

NWBoolean_ts        ccode;
uint16              serverConnID;

ccode=NWTTSIsAvailable( serverConnID );
```

Input

serverConnID	Passes the file server connection ID.

Output

None.

Return Values

1 Transaction Tracking is available. (See "Description" below.)

0 Transaction Tracking is not available. (See "Description" below.") If another problem exists, one of the following error codes is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The version of NetWare that the file server is running determines the information that this call returns.

For NetWare 2.x, this call allows the application to know whether the server has TTS installed.

1 Indicates that TTS is installed.

0 TTS is not installed.

For NetWare 2.x, the API does not indicate whether TTS is currently enabled. To ensure that TTS is enabled, NWTTSEnableTransactionTracking should be called.

For NetWare 3.x, this call indicates whether TTS is enabled.

1 Indicates that TTS is enabled.

0 Indicates that TTS has been disabled.

See Also

NWTTSEnableTransactionTracking
NWTTSDisableTransactionTracking

NWTTSIsTransactionWritten

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function verifies whether a transaction has been written to disk.

Synopsis

```
#include "nwapi.h"

NWBoolean_ts        ccode;
uint16              serverConnID;
uint32              transactionReferenceNumber;

ccode=NWTTSIsTransactionWritten( serverConnID,
        transactionReferenceNumber );
```

Input

serverConnID	Passes the file server connection ID.
transactionReferenceNumber	Passes the Transaction Reference number, obtained from the NWTTSEndTransaction function.

Output

None.

Return Values

1 Transaction written to disk
0 Transaction not written to disk, and an error code is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

Before using NWTTSIsTransactionWritten, use NWTTSEndTransaction to obtain a valid transactionReferenceNumber. If NWTTSEndTransaction fails, do not use NWTTSIsTransactionWritten. When NWTTSEndTransaction fails, it returns an invalid transactionReferenceNumber. When NWTTSIsTransactionWritten is passed an invalid transactionReferenceNumber, it returns an invalid response.

Applications should not wait for transactions to be written to disk unless it is absolutely necessary. Because of the file server caching algorithms, it may be 3 to 5 seconds (or longer) before they are actually written.

Notes

Transactions are written to disk in the order in which they terminate.

See Also

NWTTSEndTransaction

NWTTSSetConnectionThresholds

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function informs NetWare of how many explicit physical and logical record locks to permit before invoking implicit transactions.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint8       logicalRecordLockThreshold;
uint8       physicalRecordLockThreshold;

ccode=NWTTSSetConnectionThresholds( serverConnID,
        logicalRecordLockThreshold, physicalRecordLockThreshold );
```

Input

serverConnID	Passes the file server connection ID.
logicalRecordLockThreshold	Passes the number of logical record locks to allow before implicit transactions begin.
physicalRecordLockThreshold	Passes the number of physical record locks to allow before implicit transactions begin.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

This function and NWTTSGetConnectionThresholds are useful for applications that change the implicit application thresholds and later want to restore them. For example, NWTTSGetConnectionThresholds can obtain the current number of logical and physical locks and then NWTTSSetConnectionThresholds can do one of the following:

- Turn off implicit transactions. (Applications that use only explicit transactions, but sometimes generate unnecessary implicit transactions, need to turn off all implicit transactions.)
- Set implicit thresholds for applications that always keep one or more records locked.

Notes

The default threshold for logical and physical locks is 0. A threshold of 255 means no implicit transactions for that lock type will be performed.

See Also

NWTTSGetConnectionThresholds

NWTTSSetControlFlags

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
		✓

This function enables or disables automatic record locking on writes to transactional files.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint8       TTSControlFlags;

ccode=NWTTSSetControlFlags( serverConnID, TTSControlFlags );
```

Input

serverConnID	Passes the file server connection ID.
TTSControlFlags	Passes the Transaction Tracking Control flags. (See "Description" below.)

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

Transaction tracking control flags are only valid for files flagged as TTS (transactional). These flags are defined as follows:

0x00	Automatic record locking is disabled
0x01	Automatic record locking is enabled

See Also

NWTTSGetControlFlags

NWTTSSetProcessThresholds

NetWare 2.x	NetWare 3.x	NetWare for AViiON Systems
✓	✓	✓

This function sets the number of logical and physical locks to perform before implicit locking begins.

Synopsis

```
#include "nwapi.h"

int         ccode;
uint16      serverConnID;
uint8       logicalRecordLockThreshold;
uint8       physicalRecordLockThreshold;

ccode=NWTTSSetProcessThresholds( serverConnID,
        logicalRecordLockThreshold, physicalRecordLockThreshold );
```

Input

serverConnID	Passes the file server connection ID.
logicalRecordLockThreshold	Passes the number of logical record locks to allow before implicit transactions begin.
physicalRecordLockThreshold	Passes the number of physical record locks to allow before implicit transactions begin.

Output

None.

Return Values

0 Successful.

-1 Unsuccessful. One of the following error codes is placed in NWErrno:

0xFE	Transaction Restart
0xFD	Transaction Tracking Disabled
0xFF	Lock Error
0xFF	No Explicit Transaction Active

Note: See Appendix B for a complete listing of possible NetWare errors.

Description

The thresholds set by this function are valid for the requesting application only. When the application terminates, the default workstation thresholds are restored. This function is useful in either turning off implicit transactions or allowing applications that always keep one or more records locked to work. Applications that intend to use only explicit transactions, but sometimes generate unnecessary implicit transactions, can use this function to turn off all implicit transactions.

Notes

The default threshold for logical and physical locks is 0 unless this number has been changed using the NWTTSSetConnectionThresholds function. A threshold of 255 means no implicit transactions for that lock type are performed.

See Also

NWTTSGetProcessThresholds

End of Chapter

Appendix A
Constant Declarations and Structure Definitions

The following constant and structure definitions are contained in the nwapi.h include file.

Accounting Services

Constant Definitions:

NWMAX_COMMENT_LENGTH	48
NWMAX_NUMBER_OF_HOLDS	32
NWMAX_OBJECT_NAME_LENGTH	48

Comment Types:

NWAN_CONNECT_TIME	1
NWAN_DISK_STORAGE	2
NWAN_LOG_IN	3
NWAN_LOG_OUT	4
NWAN_ACCOUNT_LOCKED	5
NWAN_SERVER_TIME_MODIFIED	6

NWHoldInfo_t Structure:

```
typedef struct {
    uint32  objectID;
    int32   holdAmount;
} NWHoldInfo_t;
```

The objectID field contains the bindery objectID of the object submitting the hold.

The holdAmount field contains the amount that the objectID placed against the user's account balance.

Bindery Services

Constant Definitions:

NWMAX_MEMBER_NAME_LENGTH	48
NWMAX_OBJECT_NAME_LENGTH	48
NWMAX_PASSWORD_LENGTH	16
NWMAX_PROPERTY_NAME_LENGTH	16
NWMAX_PROPERTY_VALUE_LENGTH	128
NWMAX_SEGMENT_DATA_LENGTH	128

Bindery Object Types:

NWOT_WILD	0xFFFF
NWOT_UNKNOWN	0x0000
NWOT_USER	0x0001
NWOT_USER_GROUP	0x0002
NWOT_PRINT_QUEUE	0x0003
NWOT_FILE_SERVER	0x0004
NWOT_JOB_SERVER	0x0005
NWOT_GATEWAY	0x0006
NWOT_PRINT_SERVER	0x0007
NWOT_ARCHIVE_QUEUE	0x0008
NWOT_ARCHIVE_SERVER	0x0009
NWOT_JOB_QUEUE	0x000A
NWOT_ADMINISTRATION	0x000B
NWOT_NAS_SNA_GATEWAY	0x0021
NWOT_REMOTE_BRIDGE_SERVER	0x0024
NWOT_TIME_SYNCHRONIZATION_SERVER	0x002D
NWOT_ARCHIVE_SERVER_DYNAMIC_SAP	0x002E
NWOT_ADVERTISING_PRINT_SERVER	0x0047
NWOT_BTRIEVE_VAP	0x0050
NWOT_PRINT_QUEUE_USER	0x0053
NWOT_NVT_SERVER	0x009E

Bindery Property Types:

NWBF_ITEM	0x00	Has only one value associated with it (such as PASSWORD)
NWBF_SET	0x02	Has many values associated with it (such as GROUPS_IM_IN)

Bindery Object and Property States:

NWBF_STATIC	0x00	Permanent in bindery until deliberately deleted
NWBF_DYNAMIC	0x01	Is temporarily allocated and will be deleted when the server goes down

Bindery Object and Property Security Access Levels:

NWBS_ANY_READ	0x00	Can be read by anyone
NWBS_LOGGED_READ	0x01	Must be logged in to read
NWBS_OBJECT_READ	0x02	Can be read by same object or supervisor
NWBS_SUPER_READ	0x03	Can be read only by supervisor
NWBS_BINDERY_READ	0x04	Can be read only by the bindery
NWBS_ANY_WRITE	0x00	Can be written by anyone
NWBS_LOGGED_WRITE	0x10	Must be logged in to write
NWBS_OBJECT_WRITE	0x20	Can be written by same object or supervisor
NWBS_SUPER_WRITE	0x30	Can be written only by the supervisor
NWBS_BINDERY_WRITE	0x40	Can be written only by the bindery

NWObjectInfo_t Structure:

```
typedef struct {
    char        objectName[ NWMAX_OBJECT_NAME_LENGTH ];
    uint32      objectID;
    uint16      objectType;
    uint8       objectState;
    uint8       objectSecurity;
} NWObjectInfo_t;
```

The objectName field contains the name of the bindery object.

The objectID is the unique ID that is assigned to all bindery objects.

The objectType contains a bindery object type (see Bindery Object Types).

The objectState field contains the state assigned to the object. (See Bindery Object and Property States.)

The objectSecurity field contains the security assigned to the object. (See Bindery Object and Property Security Access Levels.)

NWPropertyInfo_t Structure:

```
typedef struct {
    char    propertyName[ NWMAX_PROPERTY_NAME_LENGTH ];
    uint8   propertyStateAndType;
    uint8   propertySecurity;
    uint8   propertyHasAValue;
} NWPropertyInfo_t;
```

The propertyName field contains the name of the bindery property. See *NetWare® for AViiON® Series Systems C Interface Programmer's Guide* for a complete listing of the assigned properties.

The propertyStateAndType field contains the state and type of the property. (See Bindery Object and Property States). The Type flag is OR'ed together with the State flag.

The propertySecurity contains the security assigned to the property. (See Bindery Object and Property Security Access Levels.)

The propertyHasAValue field contains 0 if there are no associated values or -1 (0xFF) if the property does have a value.

Connection Services

Constant Definitions:

NWMAX_CONNECTION_LIST_LENGTH	50
NWMAX_INTERNET_ADDRESS_LENGTH	12
NWMAX_KEYED_PASSWORD_LENGTH	8
NWMAX_LOGIN_TIME_LENGTH	7
NWMAX_OBJECT_NAME_LENGTH	48

File and Path Services

Constant Definitions:

NWMAX_DIR_NAME_LENGTH	16
NWMAX_DIR_PATH_LENGTH	255
NWMAX_DS_NAME	16
NWMAX_FILE_HANDLE_SIZE	6
NWMAX_FILE_NAME_LENGTH	16
NWMAX_NS_COUNT	10
NWMAX_NS_NAME	16
NWMAX_NUM_NS	10
NWMAX_NUM_DS	10
NWMAX_SERVER_NAME_LENGTH	48
NWMAX_USER_RESTRICTION	12
NWMAX_VOLUME_NAME_LENGTH	16

File Attributes

NWFA_NORMAL	0x00000000L
NWFA_READ_ONLY	0x00000001L
NWFA_HIDDEN	0x00000002L
NWFA_SYSTEM	0x00000004L
NWFA_EXECUTE_ONLY	0x00000008L
NWFA_NEED_ARCHIVE	0x00000020L
NWFA_SHARABLE	0x00000080L
NWFA_TRANSACTIONAL	0x00001000L
NWFA_INDEXED	0x00002000L
NWFA_READ_AUDIT	0x00004000L
NWFA_WRITE_AUDIT	0x00008000L
NWFA_PURGE	0x00010000L
NWFA_RENAME_INHIBIT	0x00020000L
NWFA_DELETE_INHIBIT	0x00040000L
NWFA_COPY_INHIBIT	0x00080000L

NWFA_NORMAL	The normal attribute allows read/write access to the file on both NetWare 2.x and NetWare 3.x.
NWFA_READ_ONLY	In NetWare 3.x, NetWare utilities, such as FLAG, automatically assigns the delete inhibit and rename inhibit attributes with the read only attribute. In NetWare 2.x, the additional attributes are not set, but users cannot delete or rename the file until the read only attribute is removed.
NWFA_HIDDEN	In NetWare 2.x and 3.x, this attribute hides the file from DOS DIR scans and prevents it from being deleted or copied. The files will appear using NDIR.
NWFA_SYSTEM	In NetWare 2.x and 3.x, this attribute hides the file from DOS DIR scans and prevents it from being deleted or copies. However, the files will appear using NDIR.
NWFA_EXECUTE_ONLY	In NetWare 2.x and 3.x, this attribute prevents files from being copied. Only the supervisor can assign this attribute, and it should only be assigned if the file has been backed up. Once this bit is assigned, the bit cannot be deleted. This attribute can be set with FILER.
NWFA_NEED_ARCHIVE	In NetWare 2.x and 3.x, NetWare automatically assigns this bit to files that have been modified since the last backup.
NWFA_SHARABLE	In NetWare 2.x and 3.x, this attribute allows the file to be used by more than one user at a time and is usually used in combination with the read only attribute.
NWFA_TRANSACTIONAL	In NetWare 2.x and 3.x, this attribute indicates that files will be protected by TTS. TTS ensures that, when a file is being modified, either all changes are made or no changes are made.

NWFA_INDEXED	In NetWare 3.x, the indexed attribute is no longer supported since all files are automatically turbo FAT indexed when they have 64 or more regular FAT entries and are randomly accessed. However, this attribute can still be set or cleared for use with applications. In NetWare 2.x, this attribute must be set for all files the user wants indexed. The operating system must also be configured for indexed files.
NWFA_READ_AUDIT	This attribute is associated with the NetWare Audit Trail System. The read and write audit files record information about who reads from and writes to a database file. Write audit makes continuous backup possible, and the combination of read and write audit provides greater security. **Note:** This bit is currently not supported in NetWare 2.x or 3.x. It will be in future releases.
NWFA_WRITE_AUDIT	See NWFA_READ_AUDIT. This bit is currently not supported in NetWare 2.x or 3.x. It will be in future releases.
NWFA_PURGE	In NetWare 3.x, this attribute prevents the file from being salvageable. When assigned to a file, this attribute purges the file as soon as it is deleted. NetWare 2.x does not support this attribute.
NWFA_RENAME_INHIBIT	In NetWare 3.x, this attribute restricts users from renaming files even if they have the modify right. If they have the modify right, they must remove this attribute before renaming. NetWare 2.x does not support this attribute.
NWFA_DELETE_INHIBIT	In NetWare 3.x, this attribute prevents users from erasing files even when they have been granted the erase right at the file or directory level. If they have the modify right, they can remove this attribute and then delete the file. NetWare 2.x does not support this attribute.
NWFA_COPY_INHIBIT	In NetWare 3.x, this attribute restricts only the copy rights of certain applications, such as the Macintosh Finder. If users have the modify right, they can remove the copy inhibit attribute and then copy the file.

Directory Attributes

The file attributes listed below can be assigned to directories. However, in NetWare 2.x, only NWFA_HIDDEN and NWFA_SYSTEM can be assigned to directories.

NWFA_HIDDEN	0x00000002L
NWFA_SYSTEM	0x00000004L
NWFA_PURGE	0x00010000L
NWFA_RENAME_INHIBIT	0x00020000L
NWFA_DELETE_INHIBIT	0x00040000L

NWFA_HIDDEN — This directory attribute hides the directory from DOS DIR scans and prevents it from being deleted or copied. The directory will appear using NDIR.

NWFA_SYSTEM — This directory attribute hides the directory from DOS DIR scans and prevents it from being deleted or copied. The directory will appear using NDIR.

NWFA_PURGE — This directory attribute purges all files in the directory when the files are deleted. Such files cannot be recovered or salvaged.

NWFA_RENAME_INHIBIT — This directory attribute prevents users from renaming a directory even when they have been granted the modify right. If they have the modify right, they must remove this attribute before renaming the directory.

NWFA_DELETE_INHIBIT — This directory attribute prevents the users from deleting the directory even when they have been granted erase rights to the directory. If they have the modify right, they can remove this attribute and then delete the directory.

Search Attributes:

NWSA_NONE	0x00	Normal files
NWSA_HIDDEN	0x02	Hidden and normal files
NWSA_SYSTEM	0x04	System and normal files
NWSA_BOTH	0x06	Hidden, system and normal files
NWSA_DIRECTORIES_ ONLY	0x10	Normal directories only (cannot be used on NetWare 2.x)
NWSA_FILES_ONLY	0x20	Normal files only (cannot be used on NetWare 2.x)

A file (or directory) is designated system or hidden when its corresponding file (or directory) attribute is set. The Search Attributes are used to include system and/or hidden files (or directories) in a search. In other words, if only the system bit is set in the searchAttributes parameter then all files (or directories) will be affected except hidden files (or directories). If only the hidden bit is set, all files (or directories) will be affected except system files (or directories). When neither the hidden nor the system bit is set (0x00), then only files (or directories) that are not hidden, system, or both will be affected.

Trustee Rights and Inherited Rights Mask for NetWare 3.x and NetWare for AViiON Systems.

The bit set by 0x0004 should be ignored by applications running on NetWare 3.x or NetWare for AViiON Systems. It is the "open bit" under 2.x only.

Trustee Rights apply to both files and directories in NetWare 3.x and NetWare for AViiON Systems.

See "Access Rights and Maximum Rights Mask for NetWare 2.x," below, for an explanation of NetWare 2.x rights.

NWTR_NONE	0x0000
NWTR_READ	0x0001
NWTR_WRITE	0x0002 or 0x0004 (Used only on NetWare 2.x)
NWTR_CREATE	0x0008
NWTR_ERASE	0x0010
NWTR_ACCESS	0x0020
NWTR_FILE_SCAN	0x0040
NWTR_MODIFY	0x0080
NWTR_SUPERVISOR	0x0100
NWTR_NORMAL	0x00FF
NWTR_ALL	0x01FF

Each right is described below.

NWTR_READ	For Directories: User can open and read existing files in this directory unless blocked by mask or trustee rights assignment. For Files: User can open and read this file.
NWTR_WRITE	For Directories: User can open and write to files in this directory unless blocked by mask or trustee rights assignment. For Files: User can open and write to this file.
NWTR_CREATE	For Directories: User can create files and subdirectories in this directory. For Files: User can salvage this file if it is deleted.
NWTR_ERASE	For Directories: User can delete this directory if the user has rights to delete everything inside it. For Files: User can delete this file.
NWTR_ACCESS	For Directories: User can modify the trustee list and inherited rights mask of this directory. For Files: User can modify this file's trustee list and inherited rights mask.
NWTR_FILE_SCAN	For Directories: When scanning the directory, user can see the names of files in this directory unless blocked by mask or trustee rights assignment.

	For Files: When scanning the directory, user can see the name of this file.
NWTR_MODIFY	For Directories: User can rename this directory and change the attributes of it. For Files: User can rename this file and change its attributes.
NWTR_SUPERVISOR	For Directories: User has all rights to this directory and all subdirectories and files. User can grant supervisor rights to other users in this directory and in subdirectories and files. User's rights override all inherited rights masks in subdirectories and files. User can assign space limitations to subdirectories. For Files: User has all rights to this file.
NWTR_NORMAL	For Directories: User has all of the above rights in this directory, except for NWTR_SUPERVISOR. For Files: User has all of the above rights to this file, except for NWTR_SUPERVISOR.
NWTR_ALL	For Directories: User has all of the above rights in this directory. For Files: User has all of the above rights to this file.

Trustee Access Rights and Maximum Rights Mask for NetWare 2.x

NetWare 2.x has the following rights.

NWTA_NONE	0x00
NWTA_READ	0x01
NWTA_WRITE	0x02
NWTA_OPEN	0x04
NWTA_CREATE	0x08
NWTA_DELETE	0x10
NWTA_OWNERSHIP	0x20
NWTA_SEARCH	0x40
NWTA_MODIFY	0x80
NWTA_ALL	0xFF

Each right is defined below.

NWTA_READ	If the user also has the NWTA_OPEN right, the user can open and read existing files in this directory unless blocked by the maximum rights mask.
NWTA_WRITE	If the user also has the NWTA_WRITE right, the user can open and write to files in this directory unless blocked by the maximum rights mask.
NWTA_CREATE	The user can create and salvage files in this directory unless blocked by the maximum rights mask. The Ownership right along with the Create right is required to create directories.

NWTA_DELETE	The user can delete files. The Ownership right along with the Delete right is required to delete directories. For Files: User can delete this file.
NWTA_OWNERSHIP	The user can create, rename, or delete subdirectories of the directory if the user also has the additional needed right: Create to create, Modify to rename, or Delete to delete. The user can also modify the trustee list and maximum rights mask of this directory and its subdirectories. In the NetWare utilities, this right is called the Parental right.
NWTA_SEARCH	The user can see the names of files and subdirectories in this directory unless blocked by the maximum rights mask.
NWTA_MODIFY	The user can also change the attributes of the directory, its files, and subdirectories. If the user also has the Ownership right, the user can rename the files and subdirectories.
NWTA_ALL	The user has all of the above rights in this directory.

Change Attributes:

Change attributes are used with the NWSetDirEntryInfo or NWSetFileEntryInfo function calls. These values can be OR'ed together. In these functions, you pass a structure containing the new values you want to set, and then use the following change attributes to specify which fields within the structure contain the new values.

NWCA_NAME	0x0001
NWCA_ATTRIBUTES	0x0002
NWCA_CREATE_DATE_AND_TIME	0x000C
NWCA_OWNER_ID	0x0010
NWCA_LAST_ARCHIVED_DATE_AND_TIME	0x0060
NWCA_LAST_ARCHIVED_ID	0x0080
NWCA_LAST_MODIFY_DATE_AND_TIME	0x0300
NWCA_LAST_MODIFY_ID	0x0400
NWCA_LAST_ACCESSED_DATE	0x0800
NWCA_INHERITED_RIGHTS_MASK	0x1000
NWCA_DIR_RESTRICTION	0x2000

NWCA_LAST_ACCESSED_DATE and NWCA_LAST_MODIFY_ID are not available for use with directories.

NWCA_NAME	Changes the name of the directory or file.
NWCA_ATTRIBUTES	Changes the attributes of the directory or file. See "File Attributes," below, and "Directory Attributes," below. Users must have modify rights to the file or directory to change attributes.

NWCA_CREATE_DATE_AND_TIME

Changes the date and time the file or directory was created. See "File Services" in *NetWare® for AViiON® Series Systems C Interface Programmer's Guide* for a description of this four byte field. Users must have supervisor equivalence to change the date and time.

NWCA_OWNER_ID Changes the owner of the file or directory by changing the object ID in the field. Users must have supervisor equivalence to change the owner of a file or directory.

NWCA_LAST_ARCHIVED_DATE_AND_TIME

Changes the date and time the file or directory was archived. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide* for a description of this four byte field.

NWCA_LAST_ARCHIVED_ID

Changes the object ID to the user who archived the file or directory.

NWCA_LAST_MODIFY_DATE_AND_TIME

Changes the date and time the file or directory was modified. NetWare automatically updates this information when a file or directory is modified. The user must be supervisor equivalent to change this information with an API. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide* for a description of this four byte field.

NWCA_LAST_MODIFY_ID

Changes the object ID to the user who modified the file or directory. NetWare automatically updates this field when a directory or file is changed. The user must be supervisor equivalent to change this information with an API.

NWCA_LAST_ACCESSED_DATE

Changes the date and time the file was accessed. NetWare automatically updates this information when a file or directory is accessed. The user must be supervisor equivalent to change this information with an API. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide* for a description of this four byte field.

NWCA_INHERITED_RIGHTS_MASK

Changes the trustee rights in the file's or directory's inherited rights mask. See "Trustee Rights and Inherited Rights Mask for NetWare 3.x and NetWare for AViiON Systems," in this chapter, for a list of rights that can be passed. The user must have access control rights to the file or directory to change the inherited rights mask.

NWCA_DIR_RESTRICTION
Changes the directory restrictions. A 0 clears all restrictions; a 1 restricts the directory to 4KB; a 2, to 8KB; etc.

"Open file" Access Rights:

These definitions are used with NWOpenFile. One or both of the following rights must be assigned to the accessRights parameter:

NWOR_READ	0x01	Opens the file for reading and denies read access to all other users.
NWOR_WRITE	0x02	Opens the file for writing and denies write access to all other users.

The above NWOR_READ or NWOR_WRITE bit mask may be OR'ed with either NWOR_DENY_READ and NWOR_DENY_WRITE or NWOR_COMPATIBILITY.

NWOR_DENY_READ	0x04	Doesn't allow others to read to the file while you have it open
NWOR_DENY_WRITE	0x08	Doesn't allow others to write to the file while you have it open
NWOR_COMPATIBILITY	0x10	Determines access in connection with the file's "sharable" attribute.

When the NWOR_COMPATIBILITY bit is set, the following things apply:

- NWOR_DENY_READ and NWOR_DENY_WRITE are not applicable, because they will be ignored.
- When the sharable file attribute is set, the file is treated as a sharable file, no user having exclusive access.
- When the sharable file attribute is not set, one of the following occurs:
- If NWOR_COMPATIBILITY bit is OR'ed with NWOR_READ, the file is opened with write access denied to other users.
- If NWOR_COMPATIBILITY bit is OR'ed with NWOR_WRITE or both NWOR_READ and NWOR_WRITE, the file is opened with read and write access denied to other users.

All of the above information is only applicable if the open call is successful.

The following access right is available for NetWare version 3.1 and above and may be OR'ed with any of the above values:

NWOR_SYNC_MODE	0x40	Allows "write-through" access; that is, writes directly to the disk, bypassing any caching and/or buffering.

Directory and File Handle definitions:

```
typedef    uint8      NWDirHandle_ts;
typedef    uint8      NWFileHandle_ta[NWMAX_FILE_HANDLE_SIZE];
```

Handles are values that represent a complete path and file (or directory) name as defined in the file servers file handle or directory handle index table. These handles can be used to specify a file or directory without passing a complete path name. But in order to use them, you must keep track of them since there are no "NWGetHandle" functions.

NWPath_t Structure:

```
typedef struct {
    NWDirHandle_ts      dirHandle;
    uint16              serverConnID;
    char                *pathName;
} NWPath_t;
```

The dirHandle field contains either of the following:

- A 0, when a full path will be given in the pathName field.
- A value representing an allocated directory Handle.

See NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle.

The serverConnID field is the value corresponding to the server attachment. (See NWAttachToServerPlatform.)

The pathName field contains the *address* of a path that is either 1) a full path when a 0 dirHandle is used, or 2) a relative path when an allocated dirHandle is used. (The path is relative to the directory that the dirHandle represents.)

No space is allocated for pathName. The application must allocate space for the path separately and then assign pathName the address of the previously-allocated space.

NWDirEntryInfo_t Structure:

```
typedef struct {
    uint32  attributes;
    uint32  creationDateAndTime;
    uint32  ownerID;
    uint32  archiveDateAndTime;
    uint32  archiverID;
    uint32  lastModifyDateAndTime;
    uint32  dirRestriction;
    uint16  inheritedRightsMask;
    uint8   nameSpaceID;
    char    entryName[ NWMAX_DIR_NAME_LENGTH ];
} NWDirEntryInfo_t;
```

To change these fields with NWSetDirEntryInfo, you must use the Change Attributes described below.

The attributes field for directories contains a bit mask of the directory's attributes. This value will be zero when using NWScanDirEntryInfo on a file server running NetWare 2.x. See "Directory Attributes," in this chapter, and "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide.*

The creationDateAndTime field contains the date and time the directory was created. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide.*

The ownerID field is the object ID of the directory's owner. You can use NWGetObjectName to get the name of the object.

The archiveDateAndTime field contains the date and time the directory was last archived. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide.*

The archiverID field contains the objectID of the object that archived the directory. You can use NWGetObjectName to get the name of the object.

The lastModifyDateAndTime contains the date and time the directory was last modified. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide.*

The dirRestriction field contains the number of 4K blocks available to that directory and its subdirectories. This field is set with NWSetDirRestriction. The default is 0.

The inheritedRightsMask field represents the inherited rights mask of the current directory. See "Trustee Rights and Inherited Rights Mask for NetWare 3.x and NetWare for AViiON Systems," above.

The nameSpaceID contains a 0 if the directory is a DOS directory and 1 if the directory is a Macintosh directory, or other numbers representing other name spaces, if the file server has been configured for them.

The entryName contains the directory name.

NWFileEntryInfo_t Structure:

```
typedef struct {
    uint32 attributes;
    uint32 creationDateAndTime;
    uint32 ownerID;
    uint32 archiveDateAndTime;
    uint32 archiverID;
    uint32 updateDateAndTime;
    uint32 updatorID;
    uint32 fileSize;
    uint32 lastAccessDateAndTime;
    uint16 inheritedRightsMask;
    uint8  nameSpaceID;
    char   entryName[ NWMAX_FILE_NAME_LENGTH ];
} NWFileEntryInfo_t;
```

To change these fields with NWSetFileEntryInfo, you must use the Change Attributes.

The attributes field for the NWFileEntryInfo structure contains the current file's attributes. See "File Attributes," in this chapter, and "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide.*

The creationDateAndTime field contains the date and time the file was created. The time will always be 0 for NetWare 2.x. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide.*

The ownerID field is the object ID of the file's owner. You can use NWGetObjectName to get the name of the object.

The archiveDateAndTime field contains the date and time the file was last archived. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide.*

The archiverID field contains the objectID of the object that archived the file. You can use NWGetObjectName to get the name of the object.

The updateDateAndTime field contains the objectID of the object that updated the file. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide.*

The updatorID field contains the objectID of the object that updated the file. You can use NWGetObjectName to get the name of the object.

The fileSize field contains the file size in bytes.

The lastAccessDateAndTime field contains the date and time when the file was last accessed. The time will always be 0. See "File Services" in *NetWare® for AViiON® Series Systems: C Interface Programmer's Guide.*

The inheritedRightsMask field represents the inherited rights mask of the file. See "Trustee Rights and Inherited Rights Mask for NetWare 3.x and NetWare for AViiON Systems in this chapter.

The nameSpaceID contains a 0 if the directory is a DOS directory and 1 if the directory is a Macintosh directory, or other numbers representing other name spaces, if the file server has been configured for them.

The entryName contains the file name.

NWVolUsage_t Structure:

```
typedef struct {
    uint32  totalBlocks;
    uint32  availableBlocks;
    uint32  purgableBlocks;
    uint32  notYetPurgableBlocks;
    uint32  totalDirEntries;
    uint32  availDirEntries;
    uint32  maxDirEntriesUsed;
    uint16  volNum;
    uint16  sectorsPerBlock;
    uint8   isHashed;
    uint8   isCached;
    uint8   isRemovable;
    uint8   isMounted;
    char    volName[ NWMAX_VOLUME_NAME_LENGTH ];
} NWVolUsage_t;
```

This structure contains some fields that are only pertinent to NetWare 2.x and some that are only pertinent to NetWare 3.x and NetWare for AViiON Systems.

The totalBlocks field contains the total amount of 4K blocks allocated to the volume.

The availableBlocks field contains the amount of 4K blocks not used.

Note: Your version of NetWare for AViiON Systems may not support purgableBLocks and notYetPurgableBlocks functions. Refer to the release notice accompanying your shipment for specific restrictions.

The purgableBLocks field contains the amount of blocks marked for deletion and purgeable. A valid value is only returned from servers running NetWare 3.x.

The notYetPurgableBlocks contains the amount of blocks marked for deletion, but not yet purgeable, because the file server holds deleted files for a certain amount of time, before allowing them to be purged (as set by the minimum file delete wait time console command). A valid value is only returned from servers running NetWare 3.x.

The totalDirEntries contains the total amount of directories which can be created.

The availDirEntries contains the amount of directories which can be created, based on the difference between the total amount of directories and the amount of directories already created.

The maxDirEntriesUsed contains the maximum number of directories in use at one time since the volume was created.

The volNum field contains the number assigned by the file server to each volume name (beginning with 0).

The sectorsPerBlock field contains the number of 512 sectors per block within a volume. For 3.x servers this number will always be 8. For 2.x servers this is configurable from 1-16.

The isCached field will contain 0 if volume entries are not cached and non-zero if the volume entries are cached. A valid value is only returned from servers running NetWare 2.x.

The isHashed field will contain a 0 if volume entries are not hashed, and non-zero if the volume entries are hashed. A valid value is only returned from servers running NetWare 2.x.

The isRemovable field will contain 0 if the volume is on fixed media, and a non-zero value if the volume is on a removable (mountable) medium.

The isMounted field will contain a 0 if the volume is not mounted and non-zero otherwise. A valid value is only returned from servers running NetWare 2.x.

The volName field contains the name of the volume. Maximum length is 16 characters.

NWDirRestriction_t Structure:

```
typedef struct {
    uint16 level;
    uint32 maxBlocks;
    uint32 availableBlocks;
} NWDirRestriction_t;
```

The level field refers to the distance from the directory to the root directory.

The maxBlocks field contains the maximum amount of space assigned to the directory. Blocks are in 4K units.

All directories will have a value in the maxBlocks parameter. The maxBlocks parameter will return one of the following:

0x7FFFFFFF	No restrictions have ever been set.
negative value	Restrictions were set but they have been cleared. (Use a zero in NWSetDirRestriction to clear restrictions.)
positive value	Restrictions are set, and the positive value is the maximum value.

The availableBlocks field contains the amount of space assigned to a directory minus the amount of space used by the directory and its subdirectories. Blocks are in 4K units.

To calculate the amount of space in use, simply subtract availableBlocks from maxBlocks.

NWUserRestriction_t Structure:

```
typedef struct {
    uint32 objectID;
    uint32 restriction;
} NWUserRestriction_t;
```

The objectID field contains the bindery objectID of the object corresponding to the restriction.

The restriction field is specified in 4K blocks and represents the space restrictions placed within a volume on a particular object.

NWTrusteeRights_t Structure:

```
typedef struct {
    uint32 trusteeID;
    uint16 trusteeRights;
} NWTrusteeRights_t;
```

The trusteeID field contains the bindery objectID of the trustee.

The trusteeRights field refers to the rights given to the trustee for a particular directory (or file in NetWare 3.x). (See Trustee Rights and Inherited Rights Mask)

NWSalvageableInfo_t Structure:

```
typedef struct {
  uint32   deletedDateAndTime;
  uint32   deleterID;
  uint32   attributes;
  uint32   creationDateAndTime;
  uint32   ownerID;
  uint32   archiveDateAndTime;
  uint32   archiverID;
  uint32   updateDateAndTime;
  uint32   updatorID;
  uint32   fileSize;
  uint32   inheritedRightsMask;
  uint32   lastAccessDateAndTime;
  uint8    nameSpaceID;
  char     fileName[NWMAX_FILE_NAME_LENGTH];
} NWSalvageableInfo_t;
```

Note: Your version of NetWare for AViiON Systems may not support the NWSalvageableInfo_t Structure function. Refer to the release notice accompanying your shipment for specific restrictions.

The deletedDateAndTime field contains the date and time that the file was deleted.

The deleterID field refers to the bindery object ID of the object that deleted the file.

The attributes field contains a value representing the file's set attributes.

The creationDateAndTime field contains the date and time that the file was created.

The ownerID field refers to the bindery object ID of the file's owner. You can use NWGetObjectName to get the name of the object.

The archiveDateAndTime field contains the date and time that the file was last archived.

The archiverID field contains the bindery object ID of the object that archived the file. You can use NWGetObjectName to get the name of the object.

The updateDateAndTime field contains the date and time that the file was last changed.

The updatorID field refers to the bindery object ID of the object that made changes to the file. You can use NWGetObjectName to get the name of the object.

The fileSize field contains the size of the file.

The inheritedRightsMask field contains a value representing the inherited rights owned by the file.

The lastAccessDateAndTime field contains the date and time that the file was last accessed.

The nameSpaceID field contains the name space for the file (0 for DOS, 1 for Macintosh).

The fileName field contains the name of the file.

NWDataStreamInfo_t Structure:

```
typedef struct {
  uint8 definedDataStreams;
  char  dataStreamName[NWMAX_NUM_DS][NWMAX_DS_NAME];
} NWDataStreamInfo_t;
```

The definedDataStreams field contains the number of data streams defined on the file server.

The dataStreamName field contains the names of the data streams defined on the file server.

NWNameSpaceInfo_t Structure:

```
typedef struct {
    uint8  definedNameSpaces;
    char   nameSpaceName[NWMAX_NUM_NS] [NWMAX_NS_NAME];
    uint8  nameSpaceDataStreams;
    NWDataStreamInfo_t  dataStream[NWMAX_NUM_NS];
    uint8  loadedNSCount;
    uint8  loadedNS[NWMAX_NS_COUNT];
    uint8  volumesNSCount;
    uint8  volumesNS[NWMAX_NS_COUNT];
    uint8  volumesDSCount;
    uint8  volumesDS[NWMAX_NS_COUNT];
} NWNameSpaceInfo_t;
```

The definedNameSpaces field contains the number of name spaces defined on the file server.

The nameSpaceName field contains the names of the name spaces that the file server supports.

The nameSpaceDataStreams field contains the number of data streams defined on the file server.

The dataStream field contains the names of the data streams that the file server supports.

The loadedNSCount field contains the number of name spaces actually loaded on the file server.

The loadedNS field contains an index into the defined name space table.

The volumesNSCount field contains the number of name spaces that the volume is using.

The volumesNS field contains an index into the defined name space table.

The volumesDSCount contains the number of data streams that the volume is using.

The volumesDS field contains an index into the defined data stream table.

Constant Definitions:

NWMAX_BANNER_NAME_FIELD_LENGTH	13
NWMAX_BANNER_FILE_FIELD_LENGTH	13
NWMAX_CLIENT_RECORD_LENGTH	152
NWMAX_FORM_NAME_LENGTH	16
NWMAX_HEADER_FILE_NAME_LENGTH	14
NWMAX_JOB_DESCRIPTION_LENGTH	50
NWMAX_JOB_DIR_PATH_LENGTH	80
NWMAX_JOB_FILE_NAME_LENGTH	14
NWMAX_JOB_STRUCT_SIZE	256
NWMAX_NUMBER_OF_JOB_NUMBERS	250
NWMAX_NUMBER_OF_SERVER_CONN_NUMBERS	25
NWMAX_NUMBER_OF_SERVER_OBJECT_IDS	25
NWMAX_QUEUE_JOB_TIME_SIZE	6
NWMAX_QUEUE_NAME_LENGTH	48
NWMAX_QUEUE_SUBDIR_LENGTH	119
NWMAX_SERVER_STATUS_RECORD_LENGTH	64

Queue Status Flags:

NWQS_NO_SERVER_RESTRICTIONS	0x00
NWQS_NO_MORE_JOBS	0x01
NWQS_NO_MORE_SERVER_ATTACHMENTS	0x02
NWQS_SERVERS_DISABLED	0x08

NWQueueJobStruct_t Structure:

```
typedef struct {
    uint8   clientStation;
    uint8   clientTask;
    uint32  clientID;
    uint32  targetServerID;
    uint8   targetExecutionTime[NWMAX_QUEUE_JOB_TIME_SIZE];
    uint8   jobEntryTime[NWMAX_QUEUE_JOB_TIME_SIZE];
    uint16  jobNumber;
    uint16  jobType;
    uint8   jobPosition;
    uint8   jobControlFlags;
    uint8   jobFileName[NWMAX_JOB_FILE_NAME_LENGTH];
    NWFileHandle_ta jobFileHandle;
    uint8   servicingServerStation;
    uint8   servicingServerTaskNumber;
    uint32  servicingServerIDNumber;
    uint8   jobDescription[NWMAX_JOB_DESCRIPTION_LENGTH];
    NWClientRecord_ta    queueRecord;
} NWQueueJobStruct_t;

typedef char  NWClientRecord_ta[NWMAX_CLIENT_RECORD_LENGTH];
```

Of the fields defined in the NWQueueJobStruct_t structure, the user can modify only those described below.

The targetServerID field contains the server ID of the queue server that will service the job. If this field is set to 0xFFFFFFFF, any queue server can service the job. If the specified queue server is not attached to the queue, QMS removes the job from the queue.

The targetExecutionTime field indicates the earliest time that the job can be serviced. The bytes are assigned as follows: year, month, day, hour, minute, second. If this field is set to 0xFFFFFFFFFFFF, the job will be serviced at the first opportunity.

The jobType field contains a number that identifies the type of job entry. A queue server can request specific job types from a queue.

The jobControlFlags field contains flag bits indicating the status of the job. Bits in the field are set as follows:

NWCF_OPERATOR_HOLD	0x80
NWCF_USER_HOLD	0x40
NWCF_ENTRY_OPEN	0x20
NWCF_SERVICE_RESTART	0x10
NWCF_SERVICE_AUTO_START	0x08

When the NWCF_SERVICE_AUTO_START is set, the job will be serviced after a queue server connection is broken, even if the client has not cleared the Entry Open bit. If the bit is cleared when a server connection is broken, QMS removes the job from the queue.

When the NWCF_SERVICE_RESTART is set, the job remains in the queue (in its current position) when a queue server fails. If this bit is cleared, QMS removes the job from the queue, when a server fails.

When the NWCF_ENTRY_OPEN is set, the client has not filled the associated job file. The NWCloseFileAndStartQueueJob function clears this bit, marking the job is ready for service, if the User Hold and Operator Hold bits are cleared.

When the NWCF_USER_HOLD is set, the job continues to advance in the queue, but cannot be serviced until a client or operator clears this bit.

When the NWCF_OPERATOR_HOLD is set, the job continues to advance in the queue, but cannot be serviced until the operator clears this bit.

The jobDescription field contains a null-terminated ASCII text description of the content or purpose of a job. QMS displays this text as part of the job description when users or operators examine a queue.

The queueRecord field may contain any 152-byte structure that is known to the queue server.

NWPrintRecord_t Structure:

```
typedef struct {
    uint8   versionNumber;
    uint8   tabSize;
    uint16  numCopies;
    uint16  controlFlags;
    uint16  linesPerPage;
    uint16  charsPerLine;
    char    formName[NWMAX_FORM_NAME_LENGTH];
    char    bannerNameField[NWMAX_BANNER_NAME_FIELD_LENGTH];
    char    bannerFileField[NWMAX_BANNER_FILE_FIELD_LENGTH];
    char    headerFileName[NWMAX_HEADER_FILE_NAME_LENGTH];
    char    directoryPath[NWMAX_JOB_DIR_PATH_LENGTH];
} NWPrintRecord_t;
```

The versionNumber currently contains 0.

The tabSize field contains the number of spaces tabs will be expanded to (0-18).

The numCopies field contains the number of copies that will be printed.

The controlFlags field contains one or more of the following:

NWPCF_SUPPRESS_FF	0x0008
NWPCF_NOTIFY_USER	0x0010
NWPCF_TEXT_MODE	0x0040
NWPCF_PRINT_BANNER	0x0080

When NWPCF_SUPPRESS_FF is set, the form feed is suppressed.

When NWPCF_NOTIFY_USER is set, the user is notified that the job is finished.

When NWPCF_TEXT_MODE is set, tabs are expanded and the lines per page and characters per line are ignored.

When NWPCF_PRINT_BANNER is set, a banner is printed.

The linesPerPage field refers to the number of lines on one page. The design default is 66, but a default value is not currently implemented.

The charsPerLine field contains the number of characters on one line. The design default is 132, but a default value is not currently implemented.

The formName field contains the name of the form to be used in printing.

The bannerNameField field contains the text that is printed in first box of banner - usually used for user name.

The bannerFileField field contains the text printed in second box in banner -usually used for file name.

The headerFileName contains the file name that is printed in header of banner.

The directoryPath field contains the full path name of directory, where the file resides.

Server Platform Services

Constant Definitions:

NWMAX_CONNECTION_LIST_LENGTH	50
NWMAX_COMPANY_NAME_LENGTH	80
NWMAX_COPYRIGHT_NOTICE_LENGTH	80
NWMAX_DATE_LENGTH	24
NWMAX_DESCRIPTION_LENGTH	80
NWMAX_OBJECT_NAME_LENGTH	48
NWMAX_SERVER_NAME_LENGTH	48

NWDescriptionStrings_t Structure:

```
typedef struct {
  char  companyName[ NWMAX_COMPANY_NAME_LENGTH ];
  char  revisionDescription[ NWMAX_DESCRIPTION_LENGTH ];
  char  revisionDate[ NWMAX_DATE_LENGTH ];
  char  copyrightNotice[ NWMAX_COPYRIGHT_NOTICE_LENGTH ];
} NWDescriptionStrings_t;
```

Each field in this structure contains a null termination.

The companyName parameter receives the name of the company that is providing this version of NetWare.

The revisionDescription parameter receives the NetWare version and revision description string.

The revisionDate parameter receives the revision date in the form 02/15/1988.

The copyrightNotice parameter passes a pointer to the string allocated for the copyright notice.

NWServerPlatformDateAndTime_t Structure:

```
typedef struct {
    uint8   year;
    uint8   month;
    uint8   day;
    uint8   hour;
    uint8   minute;
    uint8   second;
    uint8   dayOfWeek;
} NWServerPlatformDateAndTime_t;
```

The date and time are passed in with the following values:

year becomes 0 through 99; for example: 82=1982
month becomes 1 through 12
day becomes 1 through 31
hour becomes 0 through 23
minute becomes 0 through 59
second becomes 0 through 59
dayOfWeek becomes 0 through 6 with 0 being Sunday

NWServerPlatformInfo_t Structure:

```
typedef struct {
    uint16   majorVersion;
    uint16   minorVersion;
    uint16   revision;
    uint16   SFTLevel;
    uint16   TTSLevel;
    uint16   accountingVersion;
    uint16   VAPVersion;
    uint16   queueingVersion;
    uint16   printServerVersion;
    uint16   virtualConsoleVersion;
    uint16   securityRestrictionLevel;
    uint16   internetBridgeSupport;
    uint16   maxClientConnSupported;
    uint16   clientConnInUse;
    uint16   peakClientConnUsed;
    uint16   maxVolumes;
    char     serverName[ NWMAX_SERVER_NAME_LENGTH ];
} NWServerPlatformInfo_t;
```

The majorVersion field contains the major NetWare version number.

The minorVersion field contains the minor version (or subVersion) number.

The revision field refers to the revision level of the NetWare version number.

The SFTLevel field indicates which SFT level the file server operating system is using.

The TTSLevel field indicates which TTS level the file server operating system is using.

The accountingVersion field contains the Accounting version number.

The VAPVersion contains the VAP version number.

The queueingVersion field refers to the Queuing version number.

The printServerVersion field contains the Print Server version number.

The virtualConsoleVersion field contains the Virtual Console version number.

The securityRestrictionLevel field contains the Security Restriction version number.

The internetBridgeSupport field contains the Internet Bridge support version number.

The maxClientConnSupported field indicates the maximum number of connections the file server can support.

The clientConnInUse field contains the number of connections that are currently using the file server.

The peakClientConnUsed field indicates the maximum number of connections in use at one time.

The maxVolumes field contains the maximum allowable number of volumes. For NetWare 3.x, the maximum is 32. For NetWare for AViiON Systems, the maximum is configurable.

The serverName field contains the name of the server platform.

Synchronization Services

Constant Definitions:

NWMAX_LOGICAL_RECORD_NAME_LENGTH	80
NWMAX_SEMAPHORE_NAME_LENGTH	127

File Log Flags:

For use with log file calls:

NWFL_LOG_ONLY	0x00
NWFL_LOG_AND_LOCK	0x01

Record Log Flags:

For use with physical and logical record log calls.

NWPL_LOG_ONLY	0x00
NWPL_LOG_AND_LOCK_EXCLUSIVE	0x01
NWPL_LOG_AND_LOCK_SHAREABLE	0x03

If the low-order bit is off, then the file is only logged. If the low-order bit is on, then the file is logged and locked. The high-order bit determines whether the file is locked exclusive or locked shareable. Locked has a value of 1; exclusive, 0; and shareable, 2. Thus locked exclusive is 0x01, and locked shareable is 0x03.

Record Lock Set Flags:

For use with physical and logical record lock set calls.

NWLS_EXCLUSIVE	0x00
NWLS_SHAREABLE	0x02

End of Appendix

Appendix B
NetWare Errors

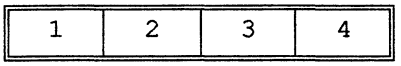
NWErrno

When a NetWare API returns an "unsuccessful" completion code (-1 for most calls, 0 for NWBoolean calls), a corresponding error code is placed in NWErrno (provided as part of NWAPI.h). NWErrno is a 4-byte value that tells you which API section the failure code came from, where the failure originated -from the client or from the file server, whether the error code is specific to one section of APIs or global among many sections, and what the actual error code is. This appendix describes the NWErrno value according to each byte, and the NetWare errors that are represented by the error code.

Note: You should declare NWErrno (in the nwapi.h include file) and include NWERRORS.H in your code if you want to receive NetWare errors. The syntax would be as follows:

```
#include "nwerrors.h"
extern uint32 NWErrno;
```

Each byte in the 4-byte NWErrno represents one part of the total error code which helps you debug your programs calling NetWare APIs. The bytes are labelled according to the following diagram.



The 1st byte in NWErrno is reserved for future use.

The 2nd byte in NWErrno identifies the area of APIs from which the error originated (Accounting, Bindery, Connection, etc.). The following values correspond to the API sections:

0x01 - Accounting
0x07 - Bindery
0x09 - Connection
0x11 - Files
0x1B - Path
0x21 - Queue
0x23 - Server Platform
0x27 - Synchronization
0x29 - Transaction Tracking
0x15 - Miscellaneous

The 3rd byte in NWErrno will identify what generated the error: the file server or an API. This byte will contain one of the following values:

0x00 - File server
0x10 - API
0x20 - API

If a file server value is returned, the call failed after a valid request was made to the file server. If an API value is returned, the call failed before a request was made to the file server.

The 4th Byte in NWErrno will contain a value which represents one of the following NetWare Errors (as defined in NWERRORS.H).

Errors returned in the 4th Byte of NWErrno:

NWERR_PATH_TOO_LONG	0x01
NWERR_BAD_SRCH_DRIVE_VECTOR	0x01
NWERR_VERSION_TOO_LOW	0x01
NWERR_INVALID_PARAMETER_LENGTH	0x01
NWERR_SETTING_UP_TRANSPORT	0x01
NWERR_INVALID_PATH_LENGTH	0x02
NWERR_SFT_LEVEL_TOO_LOW	0x02
NWERR_TRANSPORT_OPEN	0x02
NWERR_TRANSPORT_CLOSE	0x03
NWERR_NOT_CONNECTED_TO_SERVER	0x04
NWERR_CONNECT_LIST_OVERFLOW	0x04
NWERR_TTS_LEVEL_TOO_LOW	0x04
NWERR_AFP_LEVEL_TOO_LOW	0x08
NWERR_INVALID_DRIVE_NUM	0x20
NWERR_NO_MORE_RESTRICTIONS	0x24
NWERR_NO_DRIVES_AVAILABLE	0x25
NWERR_WS_DOES_NOT_SUPPORT	0x30
NWERR_INVALID_CONNECTION_ID	0x30
NWERRR_INVALID_NCP_PACKET_LENGTH	0x31
NWERR_SETTING_UP_TIMEOUT	0x32
NWERR_TRANSPORT_SEND	0x33
NWERR_IN_SETTING_SIGNALS	0x34
NWERR_FILE_IN_USE_ERROR	0x80
NWERR_NO_MORE_FILE_HANDLES	0x81
NWERR_NO_OPEN_PRIVILEGES	0x82
NWERR_IO_ERROR_NETWORK_DISK	0x83
NWERR_NO_CREATE_PRIVILEGES	0x84
NWERR_NO_CREATE_DELETE_PRIVILEGES	0x85
NWERR_CREATE_FILE_EXISTS_READ_ONLY	0x86
NWERR_CREATE_ERROR	0x87
NWERR_WILDCARDS_IN_CREATE_FILE_NAME	0x87
NWERR_INVALID_FILE_HANDLE	0x88
NWERR_NO_SEARCH_PRIVILEGES	0x89
NWERR_NO_DELETE_PRIVILEGES	0x8A
NWERR_NO_RENAME_PRIVILEGES	0x8B
NWERR_NO_MODIFY_PRIVILEGES	0x8C
NWERR_SOME_FILES_AFFECTED_IN_USE	0x8D
NWERR_ENTRY_IN_USE	0x8D
NWERR_NO_FILES_AFFECTED_IN_USE	0x8E
NWERR_SOME_FILES_AFECTED_READ_ONLY	0x8F
NWERR_NO_FILES_AFFECTED_READ_ONLY	0x90
NWERR_ALL_FILE_READ_ONLY	0x90
NWERR_SOME_FILES_RENAMED_NAME_EXISTS	0x91
NWERR_NO_FILES_RENAMED_NAME_EXISTS	0x92
NWERR_NO_READ_PRIVILEGES	0x93
NWERR_INVALID_OPEN_ACCESS_RIGHTS	0x94
NWERR_NO_WRITE_PRIVILEGES_OR_READ_ONLY	0X94
NWERR_FILE_DETACHED	0x95

NWERR_SERVER_OUT_OF_MEMORY	0x96
NWERR_NO_DISK_SPACE_FOR_SPOOL_FILE	0x97
NWERR_VOLUME_DOES_NOT_EXIST	0x98
NWERR_BAD_VOL_NUM	0x98
NWERR_DIRECTORY_FULL	0x99
NWERR_INVALID_NAME	0x99
NWERR_RENAMING_ACROSS_VOLUMES	0x9A
NWERR_BAD_DIRECTORY_HANDLE	0x9B
NWERR_INVALID_PATH	0x9C
NWERR_NO_MORE_TRUSTEES	0x9C
NWERR_NO_MORE_DIRECTORY_HANDLES	0x9D
NWERR_INVALID_FILENAME	0x9E
NWERR_DIRECTORY_ACTIVE	0x9F
NWERR_DIRECTORY_NOT_EMPTY	0xA0
NWERR_DIRECTORY_IO_ERROR	0xA1
NWERR_READ_FILE_WITH_RECORD_LOCKED	0xA2
NWERR_LOGIN_DENIED_NO_ACCOUNT_BALANC	0xC1
NWERR_CREDIT_LIMIT_EXCEEDED	0xC2
NWERR_LOGIN_DENIED_NO_CREDIT	0xC2
NWERR_INTRUDER_DETECTION_LOCK	0xC5
NWERR_LOGIN_LOCKOUT	0xC5
NWERR_NO_CONSOLE_OPERATOR_RIGHTS	0xC6
NWERR_Q_ERROR	0xD0
NWERR_NO_QUEUE	0xD1
NWERR_NO_QUEUE_SERVER	0xD2
NWERR_NO_QUEUE_RIGHTS	0xD3
NWERR_Q_FULL	0xD4
NWERR_NO_JOB	0xD5
NWERR_NO_JOB_RIGHTS	0xD6
NWERR_PASSWORD_NOT_UNIQUE	0xD7
NWERR_Q_SERVICING	0xD7
NWERR_PASSWORD_TOO_SHORT	0xD8
NWERR_Q_NOT_ACTIVE	0xD8
NWERR_LOGIN_DENIED_NO_CONNECTION	0xD9
NWERR_Q_PASSWORD_TOO_SHORT	0xD9
NWERR_UNAUTHORIZED_LOGIN_TIME	0xDA
NWERR_Q_HALTED	0xDA
NWERR_UNAUTHORIZED_LOGIN_STATION	0xDB
NWERR_MAX_QUEUE_SERVERS	0xDB
NWERR_ACCOUNT_DISABLED	0xDC
NWERR_BAD_PASSWORD	0xDE
NWERR_PASSWORD_HAS_EXPIRED_NO_GRACE	0xDE
NWERR_PASSWORD_HAS_EXPIRED	0xDF
NWERR_UNENCRYPTED_PASSWORD	0xD6
NWERR_ACCOUNT_BAD	0xD7
NWERR_NO_DISK_TRACK	0xD7
NWERR_MAX_LOGINS_EXCEEDED	0xD9
NWERR_NOT_ITEM_PROPERTY	0xE8
NWERR_WRITE_PROPERTY_TO_GROUP	0xE8
NWERR_MEMBER_ALREADY_EXISTS	0xE9
NWERR_NO_SUCH_MEMBER	0xEA
NWERR_NOT_SET_PROPERTY	0xEB
NWERR_NO_SUCH_SEGMENT	0xEC
NWERR_NO_SUCH_SET	0xEC
NWERR_PROPERTY_ALREADY_EXISTS	0xED
NWERR_OBJECT_ALREADY_EXISTS	0xEE
NWERR_INVALID_NAME	0xEF
NWERR_WILD_CARD_NOT_ALLOWED	0xF0
NWERR_ILLEGAL_WILDCARD	0xF0
NWERR_INVALID_BINDERY_SECURITY	0xF1

NWERR_NO_OBJECT_READ_PRIVILEGE	0xF2
NWERR_NO_OBJECT_RENAME_PRIVILEGE	0xF3
NWERR_NO_OBJECT_DELETE_PRIVILEGE	0xF4
NWERR_NO_OBJECT_CREATE_PRIVILEGE	0xF5
NWERR_NO_PROPERTY_DELETE_PRIVILEGE	0xF6
NWERR_NOT_SAME_LOCAL_DRIVE	0xF6
NWERR_NO_PROPERTY_CREATE_PRIVILEGE	0xF7
NWERR_TARGET_DRIVE_NOT_LOCAL	0xF7
NWERR_ALREADY_ATTACHED_TO_SERVER	0xF8
NWERR_NO_PROPERTY_WRITE_PRIVILEGE	0xF8
NWERR_NOT_ATTACHED_TO_SERVER	0xF8
NWERR_NO_FREE_CONNECTION_SLOTS	0xF9
NWERR_NO_MORE_PROP_VALS	0xF9
NWERR_NO_PROPERTY_READ_PRIVILEGE	0xF9
NWERR_NO_MORE_SERVER_SLOTS	0xFA
NWERR_TEMP_REMAP_ERROR	0xFA
NWERR_INVALID_PARAMETERS	0xFB
NWERR_NO_MORE_PROPERTIES	0xFB
NWERR_NO_SUCH_PROPERTY	0xFB
NWERR_NOT_386_FILE_SYSTEM	0xFB
NWERR_INTERNET_PACKET_REQT_CANCELED	0xFB
NWERR_SEMAPHORE_INVALID_PARMATERS	0xFB
NWERR_SYNC_INVALID_PARAMETERS	0xFB
NWERR_UNKNOWN_FILE_SERVER	0xFC
NWERR_MESSAGE_QUEUE_FULL	0xFC
NWERR_NO_SUCH_OBJECT	0xFC
NWERR_NO_MORE_OBJECTS	0xFC
NWERR_BAD_STATION_NUMBER	0xFD
NWERR_INVALID_PACKET_LENGTH	0xFD
NWERR_TTS_DISABLED	0xFD
NWERR_UNKNOWN_REQUEST	0xFD
NWERR_BINDERY_LOCKED	0xFE
NWERR_DIRECTORY_LOCKED	0xFE
NWERR_INVALID_NAME_LENGTH	0xFE
NWERR_INVALID_SEMAPHORE_NAME_LENGTH	0xFE
NWERR_IMLPICIT_TRANS_ACTIVE	0xFE
NWERR_NO_SUCH_RESTRICTION	0xFE
NWERR_PACKET_NOT_DELIVERABLE	0xFE
NWERR_RECORDS_STILL_LOCKED	0xFE
NWERR_SERVER_BINDERY_LOCKED	0xFE
NWERR_SOCKET_TABLE_FULL	0xFE
NWERR_SPOOL_DIRECTORY_ERROR	0xFE
NWERR_SUPERVISOR_HAS_DISABLED_LOGIN	0xFE
NWERR_TIMEOUT_FAILURE	0xFE
NWERR_BAD_PRINTER_ERROR	0xFF
NWERR_BAD_RECORD_OFFSET	0xFF
NWERR_CLOSE_FCB_ERROR	0xFF
NWERR_ENTRY_EXISTS	0xFF
NWERR_ENTRY_NOT_FOUND	0xFF
NWERR_EXPLICIT_TRANS_ACTIVE	0xFF
NWERR_FILE_ALREADY_EXISTS	0xFF
NWERR_FILE_EXTENSION_ERROR	0xFF
NWERR_FILE_NAME_ERROR	0xFF
NWERR_HARDWARE_FAILURE	0xFF
NWERR_INVALID_DRIVE_NUMBER	0xFF
NWERR_INVALID_INITIAL_SEMAPHORE_VALUE	0xFF
NWERR_INVALID_SEMAPHORE_HANDLE	0xFF
NWERR_IO_BOUND_ERROR	0xFF
NWERR_NO_EXPLICIT_TRANS_ACTIVE	0xFF
NWERR_NO_FILES_FOUND_ERROR	0xFF

NWERR_NO_MORE_FILES_FOUND	0xFF
NWERR_NO_RESPONSE_FROM_SERVER	0xFF
NWERR_NO_SUCH_OBJECT_OR_BAD_PASSWORD	0xFF
NWERR_OPEN_FILES	0xFF
NWERR_PATH_NOT_LOCATABLE	0xFF
NWERR_QUEUE_FULL_ERROR	0xFF
NWERR_REQUEST_NOT_OUTSTANDING	0xFF
NWERR_SEMAPHORE_INVALID_VALUE	0xFF
NWERR_SEMAPHORE_INVALID_HANDLE	0xFF
NWERR_SOCKET_ALREADY_OPEN	0xFF
NWERR_SYNC_LOCK_FAILURE	0xFF
NWERR_SYNC_ENTRY_NOT_FOUND	0xFF
NWERR_SYNC_RECORD_NOT_FOUND	0xFF
NWERR_TRANS_NOT_WRITTEN	0xFF
NWERR_TTS_NOT_AVAILABLE	0xFF

End of Appendix

Appendix C
DG/UX Errors

The NetWare APIs use two DG/UX™ interrupt signals: SIGPOLL and SIGALRM. These interrupts can cause your C calls that do kernel reads or writes to fail. If your call fails during a kernel read or write, complete the following:

1. Check the errno value. If the value is EINTR, the interrupts have caused the error.

2. Redo your read or write.

We suggest that your program check for this condition on all kernel reads and writes. The code below is an example of how you could check for the condition.

```
# include <errno.h>
...
    rvalue = read( fd, buf, cnt );
    while( rvalue == -1 ) {
        if( errno == EINTR ) {
            rvalue = read( fd, buf, cnt );
        }
        else  {
            break;
        }
    }
...
```

The interrupts can only cause the error when you have a transport open. The transport is opened using the NWAttachToServerPlatform call. Use the NWCloseTransport call to close the transport. After closing the transport, you will not have this interrupt problem.

End of Appendix

Appendix D
Differences

This Appendix describes the differences between the NetWare for AViiON Systems API Library, and the API library previously released with NetWare C Interface - DOS. It is intended for programmers who have written code to previous NetWare APIs and are modifying their code to work on a NetWare for AViiON Systems platform. The Appendix points out the key areas of change in the current APIs so that you can focus on those differences when coding to the new API library.

Use the first part of this Appendix ("Global Differences between Current and Previous APIs") to find the areas of change which you should pay particular attention to when converting previous code to the current API library.

Use the second part of this Appendix ("Function Call Index - Previous/Current API Listings") as an index to the previously released API function names (NetWare 2.x) and the corresponding NetWare for AViiON Systems API function name. The previous APIs that are not supported in this release of APIs are labelled either not supported or not in current release.

Not supported.	The APIs which are not supported will not be supported in future releases. The design and purpose of NetWare for AViiON Systems APIs prevents future support.
Not in current release.	The APIs not in the current release that can be supported and may be supported in future releases.

The index is organized according to previous APIs and the categories under which they were grouped.

API Category **Page**

New APIs provided in the current library are included under the category heading in which they are currently found. Some of the previous APIs have been moved to a new category. These are listed under both their current category and their new category.

Overview and Introduction

In general, the NetWare for AViiON Systems APIs and previously released APIs have the following differences:

Names

All current function calls now have an NW prefix on their names. All parameter names associated with structures and defined variables follow a new naming convention. Also, a few parameter and function names have been shortened or changed to more accurately reflect their function and to make them easier to use.

Types

Thirteen function calls have been modified to return boolean values. Most parameters are now defined as unsigned values. File attributes, date and time values, and client rights are now passed in larger fields than in previous API libraries.

Function

All function calls have been modified to return a separate error code along with any unsuccessful return value (while previous APIs returned an error code in place of an unsuccessful return value). Most function calls now require a serverConnID (formerly connectionNumber) to be passed in each time the call is used.

Quantity

The NetWare for AViiON Systems API library contains all of the API function calls essential to perform the most important network tasks. It does not, however, include all API function calls previously released with NetWare. Some API calls have changed categories in the documentation (directory to file services in particular) but have not changed in function.

The following API categories are currently supported:

- Accounting Services
- Bindery Services
- Connection Services
- File Services
- Path Services (formerly Directory Services)
- Queue Management Services
- Server Platform Services (formerly File Server Environment Services)
- Synchronization Services
- Transaction Tracking Services

Some calls have been replaced by more efficient calls, and some are simply not relevant to the environment under which the NetWare for AViiON Systems API calls will be used. Some of the function calls not included with the current API library worked directly through the DOS Shell, a TSR (Terminate Stay Resident) program that monitored local environment activity and kept tables of information on the network connections.

For example, the previous NetWare printing calls worked directly through the DOS shell. The printing calls would capture print jobs headed to a local printer port and re-route them to a network printer. The NetWare for AViiON Systems API calls simply send the print job directly to the network queue and thereby make the printing calls unnecessary (and undesirable because of their implementation through a DOS Shell/TSR Program which makes them non-portable across many environments).

Global Differences Between Current and Previous APIs

The following pages describe the global differences between the NetWare for AViiON Systems API library and those previously released. These differences are described in general and should be used as a guide to the specifics documented in the *API Reference Guide: NetWare C Interface for NetWare*, Revision 1.0.

Boolean Function Calls

The following function calls return boolean values (1 for true, 0 for false), rather than the return values of prior (and most of the current) NetWare APIs which return zero for true (successful) and non-zero values for false (unsuccessful). This has been done to simplify using these functions in a loop.

Bindery Services

NWScanObject
NWScanProperty
NWScanPropertyValue
NWIsObjectInSet
NWIsObjectPasswordOK

File Services

NWScanDirEntryInfo
NWScanFileEntryInfo
NWScanSalvageableFiles
NWScanTrusteePaths

Transaction Tracking Services

NWTTSIsAvailable

Connection IDs and Connection Numbers

NetWare for AViiON Systems APIs use differently named parameters to identify user and server connections. These two API parameter names have changed as follows:

connectionNumber	to	**clientConnID**
connectionID	to	**serverConnID**

The new parameter names more closely reflect the meaning of the parameter:

clientConnID The user's (client's) connection number in the file server's table.

serverConnID A unique file server in the user tables (created and tracked by the NetWare for AViiON Systems APIs).

ServerConnID is used in almost all NetWare for AViiON Systems calls.

Unlike previous APIs that used a "primary, preferred, or default" server connection ID, NetWare for AViiON Systems APIs pass one server connection ID with every call: serverConnID. The serverConnID (a uint16 value) must be kept by the user where previously this data was stored by the DOS Shell. The NWGetServerConnID function is available to retrieve a lost ID number if necessary.

Having a one to one correlation between the ID number and a server negates the need to use (and maintain) the primary, preferred, or default ID. Obviously, users can create their own algorithms if those options are desired. But using the current APIs leaves no doubt about which server a function is accessing. And passing a variable in place of serverConnID to other APIs further simplifies using the new connection parameter.

Date and time

The date and time parameters for file creation and archiving, formerly two 2-byte fields, have changed to one 4-byte field. This does not affect how the file server date and time is specified in a structure.

DOS Shell

NetWare for AViiON Systems APIs access the network directly and do not use a DOS shell (as previous APIs did) to store and transfer data. A user must now keep track of data which was previously kept by the shell (in tables).

Data which must now be kept by the user includes:

- File and Directory Handles
- Drive Mappings
- Server Connection IDs
- Print Job Routing

Without a DOS shell, some APIs become extraneous and therefore not currently supported. But in most cases where an API is not supported, an alternative exists for accessing the data necessary to perform a desired network task. For example, a directory handle can always be obtained (even without the GetDirHandle call) by simply re-allocating a handle.

The following DOS-shell-specific APIs are not currently supported:

- WorkStation Services or Environment Calls
  - GetShellVersion Information
  - GetNumberOfLocalDrives
- Directory Handles, File Handles, and Mappings
  - GetDirHandle
  - GetFileHandle
- Server Connection IDs
  - SetPrimary(Preferred, or Default)ConnectionID
  - GetPrimary(Preferred, or Default)ConnectionID
- Print Services
  - StartLPTCapture
  - SetCapturePrintJobs

The NetWare for AViiON Systems APIs provide new functions to open, read, write, and close a file (a task previously handled by the shell). Some of the new calls include:

- NWOpenFile
- NWReadFile
- NWWriteFile
- NWCloseFile

Since code written to the NetWare for AViiON Systems APIs can be run without a DOS shell, the resulting code becomes portable across any platform, with or without a shell. Users might find it easier to access and use the stored data. Overhead is reduced through the ability to write directly to queues and to use only one connectionID for each target server.

Error Codes - Return Values

NetWare for AViiON Systems APIs return a completion code representing success or failure only. Previous versions of NetWare APIs returned a 1-byte completion code representing successful completion, or a non-zero value which represented an error code (which also meant the call had failed).

The NetWare for AViiON Systems APIs return a 4-byte error code by doing the following:

- Returning a 1-byte completion code (of 1, 0, or -1) signifying successful or unsuccessful completion only.
- Placing a 4-byte error code in the NWErrno field (provided with nwapi.h) along with any unsuccessful completion code.

The 4-byte NWErrno field allows more information to be passed back and should aid in debugging code or analyzing utilities. This field identifies the location and type of returned errors as follows:

- The 4th byte used for the traditional error codes;
- The 2nd byte identifies the section (Accounting, Bindery, Connection, etc.) from which the error originated;
- The 3rd byte identifies the type of error - whether it is unique to a particular section of calls or prevalent throughout the many types of function calls;
- The 1st byte is reserved for future use.

Global Function Name Change - NW Prefix

An uppercase NW has been added to the beginning of each function call name in the NetWare for AViiON Systems API library.

Example: WritePropertyValue (DOS API call) has been changed to NWWritePropertyValue (NetWare for AViiON Systems API call)

Parameter Definitions And Structures -

NetWare for AViiON Systems APIs use structures, defined variables, and type definitions in place of lengthy parameter lists, explicit values, and frequently repeated values. All definitions are included in the nwapi.h file associated with NetWare for AViiON Systems APIs.

For example, the parameter list in the previous API named "GetFileServerDescriptionStrings" passed in four parameters, each a character string. The current API passes one parameter, NWDescriptionStrings_t, which is defined as an array of four character strings.

Current APIs have adopted the following naming convention for defined parameter types:

Prefix	Suffix	Structure Type
NW	_t	for a defined structure
NW	_ts	for a defined scalar
NW	_ta	for a defined array

Rights and Attributes

Current APIs use the NetWare 3.x security system which differs from previous NetWare versions with extended File Attributes and Trustee Rights. Previous API parameters (NetWare 2.x) which set or get rights and attributes were increased in the NetWare for AViiON Systems APIs to accommodate the larger field containing the NetWare 3.x rights and attributes. The user should read the definitions of any functions that involve security.

The new fields still accommodate NetWare 2.x rights information (if you are querying a 2.x server), but the data is returned to the user in 3.x format with the additional rights and attribute bits registering zero.

Function Call Index - Previous/Current API Libraries

The following index shows the names of previously released APIs and the current APIs which have replaced them (many have been changed in name only). This index also includes any differences which are unique to an API category, and the status of the calls in it.

This index should be used to direct you to the current API description in the *NetWare® for AViiON® Series Systems C Interface Reference Guide*, where you will find a description of the new APIs including differences in function and parameter names as well as any global changes described previously in this Appendix.

Note: This list is generated from the APIs in *NetWare C Interface - DOS* (1989, NetWare 2.x). This list might not include every API function call previously released by Novell.

Accounting Services

Accounting services are fully supported with the only differences being those described previously in this Appendix.

Accounting Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
GetAccountStatus	NWGetAccountStatus
SubmitAccountCharge	NWSubmitAccountCharge
SubmitAccountHold	NWSubmitAccountHold
SubmitAccountNote	NWSubmitAccountNote

AFP Services

AppleTalk Filing Protocol (AFP), namespace, and data stream and data fork APIs are not currently supported.

AFP Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
AFPAllocTemporaryDirHandle	Not in current release.
AFPCreateDirectory	Not in current release.
AFPCreateFIle	Not in current release.
AFPDelete	Not in current release.
AFPDirectoryEntry	Not in current release.
AFPGetEntryIDFromName	Not in current release.
AFPGetEntryIDFromNetWareHandle	Not in current release.
AFPGetEntryIDFromPathName	Not in current release.
AFPGetFileInformation	Not in current release.

(continued)

AFP Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
AFPOpenFileFork	Not in current release.
AFPRename	Not in current release.
AFPScanFileInformation	Not in current release.
AFPSetFileInformation	Not in current release.
AFPSupported	Not in current release.

(concluded)

Bindery Services

All bindery services are currently supported. In addition to any differences described previously in this Appendix, the word "bindery" has been taken out of all bindery service function names (except for NWOpenBindery, NWCloseBindery, and NWGetBinderyAccessLevel).

Bindery Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
AddBinderyObjectToSet	NWAddObjectToSet
ChangeBinderyObjectPassword	NWChangeObjectPassword
ChangeBinderyObjectSecurity	NWChangeObjectSecurity
ChangePropertySecurity	NWChangePropertySecurity
CloseBindery	NWCloseBindery
CreateBinderyObject	NWCreateObject
CreateProperty	NWCreateProperty
DeleteBinderyObject	NWDeleteObject
DeleteBinderyObjectFromSet	NWDeleteObjectFromSet
DeleteProperty	NWDeleteProperty
GetBinderyAccessLevel	NWGetBinderyAccessLevel
GetBinderyObjectID	NWGetObjectID
GetBinderyObjectName	NWGetObjectName
IsBinderyObjectInSet	NWIsObjectInSet
OpenBindery	NWOpenBindery
ReadPropertyValue	NWScanPropertyValue
RenameBinderyObject	NWRenameObject
ScanBinderyObject	NWScanObject
ScanProperty	NWScanProperty
VerifyBinderyObjectPassword	NWIsObjectPasswordOK
WritePropertyValue	NWWritePropertyValue

Communication Services

Current APIs use the services of the Transport Layer Interface (TLI) to send and receive NCP packets to and from the file server. TLI currently uses IPX as the transport protocol but is written to be portable, allowing other transport protocols to be substituted with only minor coding changes.

IPX and SPX service calls can be valuable tools for communicating between workstation that have a DOS shell. These calls are not relevant to a workstation running without a shell. Because current APIs are used without a DOS Shell, Communication Services APIs are not currently supported.

Communication Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
AES Functions	
IPXCancelEvent	Not in current release.
IPXGetIntervalMarker	Not in current release.
IPXRelinquishControl	Not in current release.
IPXScheduleIPXEvent	Not in current release.
IPX Functions	
IPXCloseSocket	Not in current release.
IPXDisconnectFromTarget	Not in current release.
IPXGetDataAddress	Not in current release.
IPXGetInternetworkAddress	Not in current release.
IPXGetLocalTarget	Not in current release.
IPXInitialize	Not in current release.
IPXListenForPacket	Not in current release.
IPXOpenSocket	Not in current release.
IPXSendPacket	Not in current release.
SPX Functions	
SPXAbortConnection	Not in current release.
SPXEstablishConnection	Not in current release.
SPXGetConnectionStatus	Not in current release.
SPXInitialize	Not in current release.
SPXListenForConnection	Not in current release.
SPXListenForSequencedPacket	Not in current release.
SPXSendSequencedPacket	Not in current release.
SPXTerminateConnection	Not in current release.

Connection Services

All Connection Services APIs are currently supported and some new have been added. Differences from prior APIs include the addition of APIs that open and close transports (NWAttachToServerPlatform and NWCloseTransport). These APIs work with the Transportation Layer Interface (TLI), sending and receiving data through the NetWare for AViiON Systems interface. In addition to any global differences described previously in this Appendix, connection service functions names use "ServerPlatform" instead of "FileServer."

Connection Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
DetachFromFileServer	NWDetachFromServerPlatform
EnterLoginArea	Not in current release.
GetConnectionInformation	NWGetConnectionInformation
GetConnectionNumber	NWGetClientConnID
GetInternetAddress	NWGetInternetAddress
GetObjectConnectionNumbers	NWGetObjectClientConnIDs
GetStationAddress	Not in current release.
LoginToFileServer	NWLoginToServerPlatform
LogoutFromFileServer	NWLogoutFromServerPlatform
Logout	Not in current release.
New Calls:	NWClearClientConnID
	NWGetServerConnID
	NWGetServerConnIDList
	NWCloseTransport
	NWRegisterTimeoutErrorFunction

Console Control Services

With NetWare for AViiON Systems function calls, there is no specified console relevant to the APIs. Any code written to the host console would have to use host platform services and therefore the following APIs are not included in the current API library.

Console Control Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
ConsoleDisplay	Not included.
ConsoleError	Not included.
ConsoleMessage	Not included.
ConsoleQuery	Not included.
ConsoleRead	Not included.

(continued)

Console Control Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
GetScreenMode	Not included.
InString	Not included.
OutString	Not included.
PrintString	Not included.
ReadKeyboard	Not included.
SetScreenMode	Not included.

Diagnostic Services

Many diagnostic services are DOS shell specific and are therefore not currently supported by NetWare for AViiON Systems.

Diagnostic Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
IPX/SPX Component Functions	
AbortSendingPackets	Not in current release.
GetIPXSPXVersion	Not in current release.
GetIPXStatistics	Not in current release.
ReturnReceivedPacketCount	Not in current release.
StartCountingPkts	Not in current release.
StartSendingPktsTimed	Not in current release.
Driver Component Functions	
GetBridgeDriverConfiguration	Not in current release.
GetBridgeDriverStatistics	Not in current release.
GetBridgeDriverStatus	Not in current release.
GetShellDriverConfiguration	Not supported.
GetShellDriverStatistics	Not supported.
Environment Component Functions	
BeginDiagnostics	Not in current release.
EndDiagnostics	Not in current release.
Shell Component Functions	
GetOSVersionInfo	Not in current release.
GetPrimaryServerNumber	Not in current release.
GetServerAddressTable	Not in current release.
GetServerNameTable	Not in current release.
GetShellAddress	Not supported.
GetShellStatistics	Not supported.
GetShellVersionInfo	Not supported.

(continued)

Diagnostic Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
Bridge Component Functions	
GetAllKnownNetworks	Not in current release.
GetAllKnownServers	Not in current release.
GetBridgeStatistics	Not in current release.
GetLocalTables	Not supported.
GetSpecificNetworkInfo	Not in current release.
GetSpecificServerInfo	Not in current release.

(concluded)

Directory Services

Directory Service calls are now called Path Services. All of previous services are supported except those that were DOS Shell specific (all essential calls are included). Some calls have been moved to File Services. Those that have been moved are designated with a [F]. In addition to any differences described previously in this Appendix, the following abbreviations are used in this category's function names: "directory" changed to "dir," and "volume" changed to "vol."

Directory Service List

Previous API (NetWare 2.x)	Corresponding API in Current Library
AddTrusteeToDirectory	NWSetTrustee [F]
AllocPermanentDirectoryHandle	NWAllocPermanentDirHandle
AllocTemporaryDirectoryHandle	NWAllocTemporaryDirHandle
CreateDirectory	NWCreateDir [F]
DeallocateDirectoryHandle	NWDeallocateDirHandle
DeleteDirectory	NWDeleteDir [F]
DeleteTrusteeFromDirectory	NWDeleteTrustee [F]
GetCurrentDirectory	Not in current release.
GetDirectoryHandles	Not in current release.
GetDirectoryPath	NWGetDirPath
GetDriveInformation	Not in current release.
GetEffectiveDirectoryRights	NWGetEffectiveRights [F]
GetSearchDriveVector	Not in current release.
GetVolumeInformation	NWGetVolUsage [F]
GetVolumeInfoWithHandle	NWGetVolInfoWithHandle [F]
GetVolumeInfoWithNumber	Not in current release.
GetVolumeName	NWGetVolName [F]
GetVolumeNumber	NWGetVolNum [F]
IsSearchDrive	Not in current release.
MapDrive	Not in current release.
ModifyMaximumRightsMask	NWSetDirsInheritedRightsMask [F]
RenameDirectory	NWRenameDir [F]

(continued)

Directory Service List

Previous API (NetWare 2.x)	Corresponding API in Current Library
RestoreDirectoryHandle	Not in current release.
SaveDirectoryHandle	Not in current release.
ScanBinderyObjectTrusteePaths	NWScanTrusteePaths [F]
ScanDirectoryForTrustees	NWGetEntrysTrustees [F]
ScanDirectoryInformation	NWScanDirEntryInfo [F]
SetDirectoryHandle	NWSetDirHandle
SetDirectoryInformation	NWSetDirEntryInfo [F]
SetDrivePath	Not in current release.
SetSearchDriveVector	Not in current release.
New Call:	NWParseFullPath

(concluded)

File Server Environment Services

File Server Environment Services has been renamed to Server Platform Services. Most of the DOS file server environment services are supported. Some are DOS Shell specific and are therefore not supported. In addition to any differences described above in this Appendix, the term "ServerPlatform" has replaced "FileServer" in all function call names. A few calls have been moved. Those that have moved to Connection Services are designated with a [C];those that have moved to File Services are designated with a [F]; and those that have moved to Transaction Tracking Services are designated with a [T].

File Server Environment Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
CheckConsolePrivileges	Not included.
CheckNetWareVersion	Not currently supported.
ClearConnectionNumber	NWClearClientConnID [C]
DisableFileServerLogin	NWDisableServerPlatformLogin
DisableTransactionTracking	NWTTSDisableTransactionTracking [T]
DownFileServer	NWDownServerPlatform
EnableFileServerLogin	NWEnableServerPlatformLogin
EnableTransactionTracking	NWEnableTransactionTracking [T]
GetBinderyObjectDiskSpaceLeft	NWGetObjectVolRestriction [F]
GetConnectionsOpenFiles	Not supported in current release.
GetConnectionsSemaphores	Not included.
GetConnectionsTaskInformation	Not included.
GetConnectionsUsageStats	Not included.
GetConnectionsUsingFile	Not supported in current release.
GetDiskCacheStats	Not supported in current release.
GetDiskChannelStats	Not included.
GetDiskUtilization	NWGetDiskUtilization

(continued)

File Server Environment Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
GetDriveMappingTable	Not included.
GetFileServerDateAndTime	NWGetServerPlatformDateAndTime
GetFileServerDescriptionStrings	NWGetServerPlatformDescriptionStrigs
GetFileServerLANIOStats	Not supported in current release.
GetFileServerLoginStatus	NWGetServerPlatformLoginStatus
GetFileServerMiscInformation	Not included.
GetFileServerName	NWGetServerPlatformName
GetFileSystemStats	Not supported in current release.
GetLANDriverConfigInfo	Not supported in current release.
GetLogicalRecordInformation	Not supported in current release
GetLogicalRecordsbyConnection	Not supported in current release.
GetLogicalRecordInformation	Not supported in current release.
GetPathFromDirectoryEntry	NWGetDirPath [P]
GetPhysicalDiskStatistics	Not supported in current release.
GetPhyscialRecordLocksByFile	Not supported in current release.
GetPhysRecLocksbyConnect AndFile	Not supported in current release.
GetSemaphoreInformation	Not supported in current release.
GetServerInformation	NWGetServerPlatformInformation
SendConsoleBroadcast	Not supported in current release.
SetFileServerDateAndTime	NWSetServerPlatformDateAndTime
TTSGetStats	Not supported in current release.
New Call:	NWIsNetWare386

File Services

Most file services are supported. Many function calls previously categorized under "Directory Services" are now included in "File Services. In addition to any differences described previously in this Appendix, you should be aware of the following :

- All parameters containing rights and attributes will be returned in NetWare 3.x format (even if you are calling a 2.x server).
- Four calls have been added - NWOpenFile, NWReadFile, NWWriteFile, and NWCloseFile - to provide read and write capabilities previously handled by the shell.
- "Volume" has been replaced by "Vol" in all function call names
- "Directory" has been replaced by "Dir" in all function call names

Previous API (NetWare 2.x)	Corresponding API in Current Library
EraseFiles	NWDeleteFile
FileServerFileCopy	NWFileCopy
GetExtendedFileAttributes	NWGetFileAttributes
PurgeAllErasedFiles	Not included.
PurgeErasedFiles	Not included.
RestoreErasedFile	Not included.
ScanFileInformation	NWScanFileEntryInfo
SetExtendedFileAttributes	NWSetFileAttributes
SetFileInformation	NWSetFileEntryInfo
New Calls: Calls that are supported only on NetWare 3.x and not on NetWare for AViiON Systems are designated [386].	NWClearObjectVolRestriction [386] NWCloseFile NWCreateFile NWCreateNewFile NWGetDirEntryInfo NWGetDirRestriction [386] NWGetNameSpaceInfo NWGetObjectVolRestriction [386] NWGetVolsObjectRestrictions [386] NWMoveEntry NWMoveFile NWOpenFile NWPurgeSalvageableFile [386] NWReadFile NWRecoverSalvageableFile [386] NWScanSalvageableFiles [386] NWSetDirRestriction [386] NWSetFilesInheritedRightsMask NWSetObjectVolRestriction [386] NWWriteFile
Calls Moved from Directory Services:	NWCreateDir NWDeleteDir NWDeleteTrustee NWGetEffectiveRights NWGetEntrysTrustees NWGetVolInfoWithHandle NWGetVolName NWGetVolNum

(continued)

File Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
Calls Moved from Directory Services:	NWGetVolUsage
	NWRenameDir
	NWScanDirEntryInfo
	NWScanTrusteePaths
	NWSetDirsInheritedRightsMask
	NWSetDirEntryInfo
	NWSetTrustee

(concluded)

Message Services

Message Service Calls are DOS Shell specific and are therefore not currently supported by NetWare for AViiON Systems.

Message Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
BroadcastToConsole	Not in current release.
CheckPipeStatus	Not in current release.
CloseMessagePipe	Not in current release.
GetBroadcastMessage	Not in current release.
GetBroadcastMode	Not in current release.
GetPersonalMessage	Not in current release.
LogNetworkMessage	Not in current release.
OpenMessagePipe	Not in current release.
SendBroadcastMessage	Not in current release.
SendPersonalMessage	Not in current release.
SetBroadcastMode	Not in current release.

Print Services

Print Services are DOS Shell Specific and are therefore not currently supported. You can print though by using Queue Management Services calls to write directly to a NetWare queue connected to a network printer.

Print Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
CancelLPTCapture	Not supported.
CancelSpecificLPTCapture	Not supported.
EndLPTCapture	Not supported.
EndSpecificLPTCapture	Not supported.
FlushLPTCapture	Not supported.
FlushSpecificLPTCapture	Not supported.
GetBannerUserName	Not in current release.
GetDefaultCaputureFlag	Not supported.
GetDefaultLocalPrinter	Not supported.
GetLPTCaptureStatus	Not supported.
GetPrinterQueue	Not in current release.
GetPrinterStatus	Not in current release.
GetSpecificCaptureFlags	Not supported.
SetBannerUserName	Not in current release.
SetCapturePrintJob	Not supported.
SetCapturePrintQueue	Not supported.
SetDefaultCaptureFlags	Not supported.
SetDefaultLocalPrinter	Not supported.
SetSpecificCaptureFlags	Not supported.
SpecifyCaptureFile	Not supported.
StartLPTCapture	Not supported.
StartSpecificLPTCapture	Not supported.

Queue Services

Queue Management service calls are fully supported by NetWare APIs. These calls can be used in place of the Print Service calls.

Queue Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
AbortServicingQueueJob	NWAbortServicingQueueJob
AttachQueueServerToQueue	NWAttachQueueServertoQueue
ChangeQueueJobEntry	NWChangeQueueJobEntry
ChangeQueueJobPosition	NWChangeQueueJobPosition
ChangeToClientRights	NWChangeToClientRights

(continued)

Queue Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
CloseFileAndAbortQueueJob	NWCloseFileAndAbortQueueJob
CloseFileAndStartQueueJob	NWCloseFileAndStartQueueJob
CreateQueue	NWCreateQueue
CreateQueueJobAndFile	NWCreateQueueFile
DestroyQueue	NWDestroyQueue
DetachQueueServerFromQueue	NWDetachQueueServerFromQueue
FinishServicingQueueJob	NWFinishServicingQueueJob
GetQueueJobFileSize	NWGetQueueJobFileSize
GetQueueJobList	NWGetQueueJobList
ReadQueueCurrentStatus	NWReadQueueCurrentStatus
ReadQueueJobEntry	NWReadQueueJobEntry
ReadQueueServerCurrentStatus	NWReadQueueServerCurrentStatus
RemoveJobFromQueue	NWRemoveJobFromQueue
RestoreQueueServerRights	NWRestoreQueueServerRights
ServiceQueueJobAndOpenFile	NWServiceQueueJob
SetQueueCurrentStatus	NWSetQueueCurrentStatus
SetQueueServerCurrentStatus	NWSetQueueServerCurrentStatus
New Calls:	NWConvertPrintStructToQueueStruct
	NWConvertQueueStructToPrintStruct

(concluded)

SAP Services

Service Advertising Protocol (SAP) calls are DOS Shell specific and are therefore not currently supported by NetWare.

SAP Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
AdvertiseService	Not in current release.
QueryServices	Not in current release.

Synchronization Services

Synchronization Services are fully supported by NetWare for AViiON Systems. In addition to any differences described previously in this Appendix, the term "FileLock" has been replaced with "File" in all function call names that previously contained "FileLock."

Synchronization Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
ClearFile	NWClearFile
ClearFileSet	NWClearFileSet
ClearLogicalRecord	NWClearLogicalRecord
ClearLogicalRecordSet	NWClearLogicalRecordSet
ClearPhysicalRecord	NWClearPhysicalRecord
ClearPhysicalRecordSet	NWClearPhysicalRecordSet
CloseSemaphore	NWCloseSemaphore
ExamineSemaphore	NWExamineSemaphore
GetLockMode	Not in current release.
LockFileSet	NWLockFileSet
LockLogicalRecordSet	NWLockLogicalRecordSet
LockPhysicalRecordSet	NWLockPhysicalRecordSet
LogFile	NWLogFile
LogLogicalRecord	NWLogLogicalRecord
LogPhysicalRecord	NWLogPhysicalRecord
OpenSemaphore	NWOpenSemaphore
ReleaseFile	NWReleaseFile
ReleaseFileSet	NWReleaseFileSet
ReleaseLogicalRecord	NWReleaseLogicalRecord
ReleaseLogicalRecordSet	NWReleaseLogicalRecordSet
ReleasePhysicalRecord	NWReleasePhysicalRecord
ReleasePhysicalRecordSet	NWReleasePhysicalRecordSet
SetLockMode	Not in current release.
SignalSemaphore	NWSignalSemaphore
WaitOnSemaphore	NWWaitOnSemaphore

TTS Calls

Transaction Tracking Services are completely supported by NetWare APIs.

TTS Calls List

Previous API (NetWare 2.x)	Corresponding API in Current Library
TTSAbortTransaction	NWTTSAbortTransaction
TTSBeginTransaction	NWTTSBeginTransaction
TTSEndTransaction	NWTTSEndTransaction

(continued)

TTS Calls List

Previous API (NetWare 2.x)	Corresponding API in Current Library
TTSGetApplicationThresholds	NWTTSGetProcessThresholds
TTSGetWorkstationThresholds	NWTTSGetConnectionThresholds
TTSIsAvailable	NWTTSIsAvailable
TTSSetApplicationThresholds	NWTTSSetProcessThresholds
TTSSetWorkstationThresholds	NWTTSSetConnectionThresholds
TTSTransactionStatus	NWTTSIsTransactionWritten
New Calls:	NWTTSGetControlFlags
	NWTTSSetControlFlags
Calls Moved from File Server Environment Services:	NWDisableTransactionTracking
	NWEnableTransactionTracking

(concluded)

VAP Services

Value Added Process (VAP) Services are not supported in NetWare 3.x and NetWare for AViiON Systems.

VAP Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
AllocateSegment	Not supported.
CalculateAbsoluteAddress	Not supported.
ChangeProcess	Not supported.
ChangeSegmentToData	Not supported.
CreateProcess	Not supported.
DeclareExtendedSegment	Not supported.
DeclareSegmentAsData	Not supported.
DelayProcess	Not supported.
DoConsoleError	Not supported.
GetInterruptVector	Not supported.
GetProcessID	Not supported.
GetVAPHeader	Not supported.
InitializationComplete	Not supported.
KillProcess	Not supported.
SegmentToPointer	Not supported.
SetExternalProcessError	Not supported.
SetHardwareInterruptVector	Not supported.
SetInterruptVector	Not supported.
ShellPassThroughEnable	Not supported.
SleepProcess	Not supported.

(continued)

VAP Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
SpawnProcess	Not supported.
VAPAttachToFileServer	Not supported.
VAPGetConnectionID	Not supported.
VAPGetfileServerName	Not supported.
WakeUpProcess	Not supported.

(concluded)

Workstation Environment Services

Most Workstation Services are DOS Shell Specific and are therefore not supported by NetWare APIs.

Workstation Environment Services List

Previous API (NetWare 2.x)	Corresponding API in Current Library
EndOfJob	Not in current release.
GetConnectionID	NWGetServerConnID
GetDefaultConnectionID	Not supported.
GetFileServerName	NWGetServerPlatformName
GetNumberOfLocalDrives	Not supported.
GetPreferredConnectionID	Not supported.
GetPrimaryConnectionID	Not supported.
GetShellVersionInformation	Not supported.
IsConnectionIDInUse	Not in current release.
SetEndOfJobStatus	Not in current release.
SetNewWareErrorMode	Not in current release.
SetPreferredConnectionID	Not supported.
SetPrimaryConnectionID	Not supported.

End of Appendix

TIPS ORDERING PROCEDURES

TO ORDER

1. An order can be placed with the TIPS group in two ways:
   a) MAIL ORDER – Use the order form on the opposite page and fill in all requested information. Be sure to include shipping charges and local sales tax. If applicable, write in your tax exempt number in the space provided on the order form.

      Send your order form with payment to:
      Data General Corporation
      ATTN: Educational Services/TIPS G155
      4400 Computer Drive
      Westboro, MA 01581-9973

   b) TELEPHONE – Call TIPS at (508) 870-1600 for all orders that will be charged by credit card or paid for by purchase orders over $50.00. Operators are available from 8:30 AM to 5:00 PM EST.

METHOD OF PAYMENT

2. As a customer, you have several payment options:
   a) Purchase Order – Minimum of $50. If ordering by mail, a hard copy of the purchase order must accompany order.
   b) Check or Money Order – Make payable to Data General Corporation.
   c) Credit Card – A minimum order of $20 is required for Mastercard or Visa orders.

SHIPPING

3. To determine the charge for UPS shipping and handling, check the total quantity of units in your order and refer to the following chart:

Total Quantity	Shipping & Handling Charge
1-4 Units	$5.00
5-10 Units	$8.00
11-40 Units	$10.00
41-200 Units	$30.00
Over 200 Units	$100.00

   If overnight or second day shipment is desired, this information should be indicated on the order form. A separate charge will be determined at time of shipment and added to your bill.

VOLUME DISCOUNTS

4. The TIPS discount schedule is based upon the total value of the order.

Order Amount	Discount
$1-$149.99	0%
$150-$499.99	10%
Over $500	20%

TERMS AND CONDITIONS

5. Read the TIPS terms and conditions on the reverse side of the order form carefully. These must be adhered to at all times.

DELIVERY

6. Allow at least two weeks for delivery.

RETURNS

7. Items ordered through the TIPS catalog may not be returned for credit.
8. Order discrepancies must be reported within 15 days of shipment date. Contact your TIPS Administrator at (508) 870-1600 to notify the TIPS department of any problems.

INTERNATIONAL ORDERS

9. Customers outside of the United States must obtain documentation from their local Data General Subsidiary or Representative. Any TIPS orders received by Data General U.S. Headquarters will be forwarded to the appropriate DG Subsidiary or Representative for processing.

TIPS ORDER FORM

Mail To: Data General Corporation
Attn: Educational Services/TIPS G155
4400 Computer Drive
Westboro, MA 01581 - 9973

BILL TO:	SHIP TO: (No P.O. Boxes - Complete Only If Different Address)
COMPANY NAME _____	COMPANY NAME _____
ATTN: _____	ATTN: _____
ADDRESS _____	ADDRESS (NO PO BOXES) _____
CITY _____	CITY _____
STATE _____ ZIP _____	STATE _____ ZIP _____

Priority Code _____ (See label on back of catalog)

_____ _____ _____ _____ _____
Authorized Signature of Buyer (Agrees to terms & conditions on reverse side) | Title | Date | Phone (Area Code) | Ext.

ORDER #	QTY	DESCRIPTION	UNIT PRICE	TOTAL PRICE

A SHIPPING & HANDLING

☐ UPS	ADD
1-4 Items	$ 5.00
5-10 Items	$ 8.00
11-40 Items	$ 10.00
41-200 Items	$ 30.00
200+ Items	$100.00

Check for faster delivery

Additional charge to be determined at time of shipment and added to your bill.

☐ UPS Blue Label (2 day shipping)
☐ Red Label (overnight shipping)

B VOLUME DISCOUNTS

Order Amount	Save
$0 - $149.99	0%
$150 - $499.99	10%
Over $500.00	20%

Tax Exempt # or Sales Tax (if applicable) _____

ORDER TOTAL	
Less Discount See B	-
SUB TOTAL	
Your local* sales tax	+
Shipping and handling - See A	+
TOTAL - See C	

C PAYMENT METHOD

☐ Purchase Order Attached ($50 minimum)
P.O. number is _____. (Include hardcopy P.O.)
☐ Check or Money Order Enclosed
☐ Visa ☐ MasterCard ($20 minimum on credit cards)

Account Number ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ Expiration Date ☐☐☐☐

_____
Authorized Signature
(Credit card orders without signature and expiration date cannot be processed.)

THANK YOU FOR YOUR ORDER

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
PLEASE ALLOW 2 WEEKS FOR DELIVERY.
NO REFUNDS NO RETURNS.

* Data General is required by law to collect applicable sales or use tax on all purchases shipped to states where DG maintains a place of business, which covers all 50 states. Please include your local taxes when determining the total value of your order. If you are uncertain about the correct tax amount, please call 508-870-1600.

Form 702
Rev. 8/87

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form. These terms and conditions apply to all orders, telephone, telex, or mail. By accepting these products the Customer accepts and agrees to be bound by these terms and conditions.

1. CUSTOMER CERTIFICATION

Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

2. TAXES

Customer shall be responsible for all taxes, including taxes paid or payable by DGC for products or services supplied under this Agreement, exclusive of taxes based on DGC's net income, unless Customer provides written proof of exemption.

3. DATA AND PROPRIETARY RIGHTS

Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

4. LIMITED MEDIA WARRANTY

DGC warrants the CLI Macros media, provided by DGC to the Customer under this Agreement, against physical defects for a period of ninety (90) days from the date of shipment by DGC. DGC will replace defective media at no charge to you, provided it is returned postage prepaid to DGC within the ninety (90) day warranty period. This shall be your exclusive remedy and DGC's sole obligation and liability for defective media. This limited media warranty does not apply if the media has been damaged by accident, abuse or misuse.

5. DISCLAIMER OF WARRANTY

EXCEPT FOR THE LIMITED MEDIA WARRANTY NOTED ABOVE, DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS, CLI MACROS OR MATERIALS SUPPLIED HEREUNDER.

6. LIMITATION OF LIABILITY

A. CUSTOMER AGREES THAT DGC'S LIABILITY, IF ANY, FOR DAMAGES, INCLUDING BUT NOT LIMITED TO LIABILITY ARISING OUT OF CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR WARRANTY SHALL NOT EXCEED THE CHARGES PAID BY CUSTOMER FOR THE PARTICULAR PUBLICATION OR CLI MACRO INVOLVED. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO CLAIMS FOR PERSONAL INJURY CAUSED SOLELY BY DGC'S NEGLIGENCE. OTHER THAN THE CHARGES REFERENCED HEREIN, IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOST PROFITS AND DAMAGES RESULTING FROM LOSS OF USE, OR LOST DATA, OR DELIVERY DELAYS, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY THEREOF; OR FOR ANY CLAIM BY ANY THIRD PARTY.

B. ANY ACTION AGAINST DGC MUST BE COMMENCED WITHIN ONE (1) YEAR AFTER THE CAUSE OF ACTION ACCRUES.

7. GENERAL

A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts, excluding its conflict of law rules. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer. DGC hereby rejects all such different, conflicting, or additional terms.

8. IMPORTANT NOTICE REGARDING AOS/VS INTERNALS SERIES (ORDER #1865 & #1875)

Customer understands that information and material presented in the AOS/VS Internals Series documents may be specific to a particular revision of the product. Consequently user programs or systems based on this information and material may be revision-locked and may not function properly with prior or future revisions of the product. Therefore, Data General makes no representations as to the utility of this information and material beyond the current revision level which is the subject of the manual. Any use thereof by you or your company is at your own risk. Data General disclaims any liability arising from any such use and I and my company (Customer) hold Data General completely harmless therefrom.

NetWare® for AViiON® Systems: C Interface Reference Guide

069–000567–00

Cut here and insert in binder spine pocket