Bindery Services APIs

Introduction to Bindery Services

Each NetWare file server includes a small database or bindery implemented as hidden files. These hidden bindery files (NET$OBJ.SYS, NET$VAL.SYS and NET$PROP.SYS) 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 and so on. The following is a list of common 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
Advertising 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 is 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 to the chart on the next page for the values defined for each nibble.

TABLE 1. Security Levels

Hex	Binary	Access	Description
0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the server
1	0001	Logged	Access allowed only to clients who have logged in to the server
2	0010	Object	Access allowed only to clients who have logged in to the 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 with security equivalence
4	0100	NetWare	Access allowed only to NetWare

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 four 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 values 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.

TABLE 2. Create Properties

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

TABLE 3. Verify Written Values

Step	Type	API
Read value of property	Set Item	NWIsObjectInSet NWScanPropertyValue

TABLE 4. Delete Property Values

Step	Type	API
Delete a property value	Set Item	NWDeleteObjectFromSet NWWritePropertyValue

TABLE 5. Delete Properties

Step	Type	API
Delete a property Item	Set	NWDeleteProperty

NWAddObjectToSet

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

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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, ªBindery Object Types.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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

This function changes the password of a bindery object.

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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 new password.

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 cannot 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

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

Synopsis

#include ªnwapi.hº

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

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

Input

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, ªBindery Object Types.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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. The following chart describes this security level.

TABLE 6. Object Security Levels

Hex	Binary	Access	Description

0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the server
1	0001	Logged	Access allowed only to clients who have logged in to the server
2	0010	Object	Access allowed only to clients who have logged in to the 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 supervisor or as an object with supervisor security equivalence
4	0100	NetWare	Access allowed only to NetWare

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

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.

Synopsis

#include ªnwapi.hº

```
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, [a]Bindery Object Types.[o])

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

Description

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

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 determines which clients can read the property. Write security determines which clients can write to the property. Below is a chart which describes each of the security levels.

TABLE 7. Property Security Levels

Hex	Binary	Access	Description
0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the server
1	0001	Logged	Access allowed only to clients who have logged in to the server
2	0010	Object	Access allowed only to clients who have logged in to the 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 supervisor or as an object with supervisor security equivalence

4	0100	NetWare	Access allowed only to NetWare

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

This function closes the bindery on the file server associated with the file server connection ID.

Synopsis

#include ªnwapi.hº

```
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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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

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

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 ªnwapi.hº

```
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, ªBindery Object Types.º)

newObjectState passes a flag indicating the object state. (See Appendix A, ªBindery Object and Property States.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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 parameter 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 the following chart.

TABLE 8. Read and Write Values

Hex	Binary	Access	Description
0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the server
1	0001	Logged	Access allowed only to clients who have logged in to the server
2	0010	Object	Access allowed only to clients who have logged in to the 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 supervisor or as an object with supervisor security equivalence
4	0100	NetWare	Access allowed only to NetWare

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

This function adds a property to a bindery object.

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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, ªBindery Property Types and Bindery Object and Property States.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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.

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 newPropertySecurity are described in the chart on the next page.

For example, a property with newPropertySecurity 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 cannot 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

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

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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

This function deletes a member from a bindery property of type Set on the file server associated with the file server connection ID.

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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, ªBindery Object Types.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

Description

There are two types of bindery properties: Item and Set. Set properties are those that contain multiple bindery objects.

See Also

NWAddObjectToSet
NWDeleteObject
NWDeleteProperty

NWDeleteProperty

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

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

Description

The objectName and objectType parameters 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 propertyName contains wildcard characters.

Notes

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

See Also

NWDeleteObjectFromSet

NWGetBinderyAccessLevel

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

Synopsis

#include ªnwapi.hº

```
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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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. The chart below summarizes the security values.

TABLE 9. Security Values

Hex	Binary	Access	Description
0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the server
1	0001	Logged	Access allowed only to clients who have logged in to the server
2	0010	Object	Access allowed only to clients who have logged in to the 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 supervisor or as an object with supervisor security equivalence
4	0100	NetWare	Access allowed only to NetWare

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

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.

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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

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 ªnwapi.hº

```
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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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

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

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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, ªBindery Object Types.º)

Output

ccode 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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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:

TABLE 10. Bindery objects and properties

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

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

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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 error codes is placed in NWErrno:

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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

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

Synopsis

#include ªnwapi.hº

```
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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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

This function renames an object in the bindery.

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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
0xFC	No Such Object

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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

This function searches for a bindery object name.

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

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, ªNWObjectInfo_t Structure.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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.

TABLE 11. Security Options

Hex	Binary	Access	Description
0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the server
1	0001	Logged	Access allowed only to clients who have logged in to the server
2	0010	Object	Access allowed only to clients who have logged in to the 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 supervisor or as an object with supervisor security equivalence
4	0100	NetWare	Access allowed only to NetWare

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

This function searches for properties in a bindery object.

Synopsis

#include ªnwapi.hº

```
NWBoolean_ts        ccode;
uint16              serverConnID;
char                objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16              objectType;
char                searchPropertyName[NWMAX_PROPERTY_
                                NAME_LENGTH];
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, ªBindery Object Types.º)

searchPropertyName passes a pointer to the property name (with possible wildcards) being searched for.

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, ªNWPropertyInfo_t Structure.º)

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, ªNWPropertyInfo_t Structure.º)

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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
OR'ed 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. The chart below summarizes the security values.

TABLE 12. Security Values

Hex	Binary	Access	Description
0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the server
1	0001	Logged	Access allowed only to clients who have logged in to the server
2	0010	Object	Access allowed only to clients who have logged in to the 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 supervisor or as an object with supervisor security equivalence
4	0100	NetWare	Access allowed only to NetWare

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

This function reads the property value of a bindery object.

Synopsis

#include ªnwapi.hº

```
NWBoolean_ts        ccode;
uint16              serverConnID;
char                objectName[NWMAX_OBJECT_NAME_LENGTH];
uint16              objectType;
char                propertyName[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, ªBindery Object Types.º)

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 ªDescriptionº below.)

segmentData passes a pointer to the buffer allocated for the property data.

moreSegments passes a pointer to the space allocated for the ªmore segmentsº 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 ªDescriptionº 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, ªBindery Property Types.º)

Return Values

1	Object successfully found.
0	Object not found. One of the following error codes may be placed in NWErrno:

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

See Appendix B for a complete listing of possible NetWare errors and a description of the four bytes in NWErrno.

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

This function writes the property value of a bindery object.

Synopsis

#include ªnwapi.hº

```
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, ªBindery Object Types.º)

propertyName passes a pointer to the property name (type Item).

segmentNumber passes the segment number of the written data (128-byte chunks). (See ªDescriptionº on the next page.)

dataBuffer passes a pointer to the 128-byte buffer that contains the data. (See ªDescriptionº on the next page.)

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

See Appendix B for a complete list of NetWare errors and a description of the four bytes in NWErrno.

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.

You must 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