# Connection Service Group

# Connection Overview

Both Connection Services and Connection Number and Task Management Services provide functions to attach, authenticate, and license to a server. However, Connection Services are cross-platform functions (server and client), and Connection Number and Task Management Services are specific to NLM programming (server only). Use Connection Services whenever possible to provide optimal portability and functionality.

Connection: Guides

Connection Number and Task Management: Guides

# Connection

# Connection:  Guides

## Connection:  Task Guide

Attaching to Servers and Opening Connections

Getting Connection Status

Setting Connection Status

Closing and Clearing Connections

Listing Connection Handles

Manipulating Connection Numbers

Getting Connection Status Information

**Examples**

Licensing / Unlicensing a connection:  Example, Listing connection numbers:  Example, Listing Internetwork Addresses:  Example, and , Opening a Connection Using a Server Name:  Example

## Connection:  Concept Guide

Connection States

Open/Close Connection Model

Connection Handles Compared to Connection References

Connection Management Support Routines

Open and Close Functions

Connection Information Getting Functions

Connection Parameter Setting Functions

Connection Table Functions

# Connection:  Tasks

## Attaching to Servers and Opening Connections

You need not be concerned with actually attaching to a given server. The client/library manages these connections; you must simply open a connection and get a connection handle. The client makes the attachment if required and maintains a use count on each connection, ensuring that a connection is not closed if existing connection handles are still active.

The following functions return connection handles:

**NWCCOpenConnByAddr** creates a service connection using the server address.

**NWCCOpenConnByName** resolves the server name to an address, then creates a service connection.

**NWCCOpenConnByRef** opens a connection using a connection reference.

Each of these functions has an *openState* parameter to specify if the connection is licensed or unlicensed. *openState* can have one of the following values:

NWCC_OPEN_LICENSED
NWCC_OPEN_UNLICENSED

**NOTE:** Getting a connection handle does not give you rights to a server as a particular user. You must authenticate to the server as a user by calling **NWLoginToFileServer** if logging in to a bindery or **NWDSAuthenticate** if logging in to NDS.

**Related Topics:**

Connection:  Guides

Opening a Connection Using a Server Name:  Example

## Getting Connection Status

The following functions get server connection information:

**NWCCGetConnInfo**

**NWCCGetAllConnInfo**

**NWCCGetAllConnInfo**

**NWCCGetConnAddress**

**NWCCGetConnRefInfo**

**NWCCGetAllConnRefInfo**

**NWCCGetConnRefAddress**

All existing workstation connections can be scanned, including matching information for all existing connections. This is done using **NWCCScanConnRefs**.

The following functions retrieve connection information:

NWCCGetPrefServerName

**NWCCGetPrimConnRef**

**Parent Topic:**

Connection: Guides

# Setting Connection Status

If you don't require a licensed connection, conserve resources by using an unlicensed connection.

The following functions are used to change the state of authenticated connections:

**NWCCLicenseConn**

**NWCCUnlicenseConn**

The connection is unlicensed only if there are no open handles that require a licensed connection.

The following functions set connections to special cases:

**NWCCMakeConnPermanent**

A permanent connection prohibits the client from terminating this connection even after the application has terminated. Permanence defined as "not terminating after process is term." can still be closed but must be accomplished with **NWCCSysCloseConnRef**.

**NWCCSetPrefServerName**

**NWCCSetPrimConn**

**Related Topics:**

Connection: Guides

Licensing / Unlicensing a connection: Example

# Closing and Clearing Connections

As with opening connections, you need not be concerned with detaching from a server. The client is responsible for detaching connections intelligently. The client can keep a connection active for use by the same or another application. This eliminates the overhead associated with making an actual attachment.

However, if additional resources are required to open a connection to another server, the client/library can determine which connections are no longer needed and which is best to detach. It is very important that every time a connection is opened it be closed when no longer needed. The following functions are used to close connections:

**NWCCCloseConn**

**NWCCSysCloseConnRef**

**NWCCSysCloseConnRef** forces a server connection detach (used in conjunction with **NWCCMakeConnPermanent**).

> **IMPORTANT:** Use caution when calling **NWCCSysCloseConnRef** because other applications on the workstation might be using the connection.

**Parent Topic:**

Connection: Guides

# Listing Connection Handles

You can obtain a list of allocated connection handles by calling **NWCCScanConnRefs** followed by **NWCCOpenConnByRef**. These functions take a buffer and buffer size as input and returns an array of connection handles. Since the Requester allows users to configure the maximum number of connections, the size of this array can vary. Call **NWCCGetNumConns** to find the maximum number of connections supported.

Other connection handle functions include the following:

**NWCCScanConnRefs**followed by the **NWCCOpenConnByRef**

> **NOTE:** Use connection handles only as parameters in Connection Services functions. They should not be used to access the Connection Table directly.

**Related Topics:**

Connection: Guides

Listing connection numbers: Example

# Manipulating Connection Numbers

The server identifies your connection by a connection number, much the same way the Requester uses connection handles. The connection number is important to you when you need to view things from the server.

The following functions operate on connection numbers:

**NWCCGetConnInfo**

**NWClearConnectionNumber**

**NWGetObjectConnectionNumbers**

**Parent Topic:**

Connection: Guides

# Getting Connection Status Information

The**NWGetConnectionStatus**function returns connection information for either the Requester or NETX, whichever is present. This function takes a connection handle as input and returns CONNECT_INFO. Among the information returned is the following:

Connection flags

Session ID

Connection number

Server name, address, and type

Client name and type

The connection flags are a bitmap that describes the status of the connection, including whether the connection supports NDS and is authenticated.

When calling **NWGetConnectionInformation** you must be supervisor equivalent to read from or write to the bindery.

**Parent Topic:**

Connection: Guides

# Connection:  Concepts

## Connection States

A workstation can have one of three types of connections to a server:

Attached

Authenticated

Licensed

A workstation that is merely attached to a server is not granted any rights to access server resources. However, limited access to some things, such as the login directory or the ability to scan for other server addresses, are available.

After a workstation has attached to a server, it can authenticate the connection for a specific user. Authentication is the process of securely identifying the user to the server. After authenticating the connection the user is granted specific user rights to resources on the server. Directory Services enables certain aspects of the authentication process to occur without the user's knowledge.

After the user has an authenticated connection to the Directory tree, authentication can occur to any server in the tree without requiring a password. This is known as background authentication.

Licensed connections enable the use of mapping, file system and printing functions.

**Parent Topic:**

Connection:  Guides

## Open/Close Connection Model

The model used to manage all connections is an open/close model. Many NetWare functions contain a connection handle parameter. Connection handles should be considered as limited resources: get a connection handle when you need it and release it when you are done with it.

When you need a connection handle you must open a connection to a server. A connection handle is returned. The call to open a connection might or might not establish a connection. If there is no current connection to the server and a connection is established, a connection handle is returned.

However, if a connection to the server already exists, a unique connection handle is returned (and the client/library notes that two connection handles exist for that server).

> **NOTE:** Do not copy or duplicate connection handles. Also, you cannot compare connection handles to determine if they are to the same server.

When you are finished using the connection handle, the handle must be closed. Closing the handle notifies the client/library that this handle is no longer needed. If other connection handles are open to the same server (either by this application or by another application on the user's workstation) the connection is not closed. The client/library notes that one less connection handle is opened to that server.

Once a connection handle is closed, it is invalid and cannot be reused. If you need a connection to the same server again, a new connection handle must be obtained by opening the connection and getting a new connection handle. This open/close model for connection handles allows the workstation client to intelligently manage server connections. You can now safely close a connection without having to worry if another application needs the connection.

When a connection to a server is no longer required by the applications on a user's workstation, the connection might not actually be closed. The client/library notes that no application is using the connection and it is made available for use either to connect to the same server or to another server.

**Parent Topic:**

Connection: Guides

# Connection Handles Compared to Connection References

Connection references allow you to maintain a reference to a connection without having a connection handle to the connection. This is useful when you need to open and close connections frequently. The reference to the connection makes getting the connection much faster. However, when you do not have a connection handle, the client/library can close the connection without your knowledge.

References are returned by calling **NWCCScanConnRefs**. The reference cannot be used in place of the connection handle. However, by calling **NWCCOpenConnByRef**, a referenced connection to a server can be opened and a valid connection handle returned. As long as the client/library does not close the connection, the new connection can be opened more quickly than by getting a connection without a reference. Given a connection handle, a connection reference can be obtained by calling **NWCCGetConnRef**.

**Parent Topic:**

Connection: Guides

# Connection Management Support Routines

The nwconnec.h header declares a group of functions that perform NETX-style connection operations. These functions rely on bindery-oriented information (bindery object names and IDs). Using them assures compatibility with both the NetWare Requester™ and NETX. The functions do not support Directory Services.

**Parent Topic:**

Connection: Guides

# Open and Close Functions

| Function | Description |
|---|---|
| **NWCCCloseConn** | Closes the specified connection. |
| **NWCCLicenseConn** | Licenses the specified connection. |
| **NWCCOpenConnByAddr** | Opens a connection using a network address. |
| **NWCCOpenConnByName** | Resolves the given name to a network address then creates a connection to that address. |
| **NWCCOpenConnByPref** | Opens an initial connection using the configured preferred settings. |
| **NWCCOpenConnByRef** | Opens a connection associated with the given connection reference. |
| **NWCCSysCloseConnRef** | Closes and detaches the specified connection, including the connection reference and all connection handles for this connection. |
| **NWCCUnlicenseConn** | Unlicenses the specified licensed connection. |

**Parent Topic:**

Connection: Guides

# Connection Information Getting Functions

| Function | Comment |
|---|---|
| **NWCCGetAllConnInfo** | Returns all information for the specified connection. |
| **NWCCGetAllConnRefInfo** | Returns all information for a specified connection reference. |
| **NWCCGetConnAddress** | Returns the transport address for the specified connection. |
| **NWCCGetConnAddressLength** | Returns the length of the connection address for the specified connection. |
| **NWCCGetConnInfo** | Returns information about the specified connection. |
| **NWCCGetConnRef** | Returns the connection reference for the specified connection. |
| **NWCCGetConnRefAddress** | Returns the transport address for the specified connection reference. |
| **NWCCGetConnRefAddressLength** | Returns the length of the connection address for the specified connection reference. |
| **NWCCGetConnRefInfo** | Returns the specified information for a given connection reference. |
| **NWCCGetPrefServerName** | Returns the name from the *PREFERRED SERVER* parameter. |
| **NWCCGetPrimConnRef** | Returns the primary connection reference. |
| **NWCCScanConnRefs** | Returns a connection reference for each connection on the workstation. |

**Parent Topic:**

Connection: Guides

# Connection Parameter Setting Functions

| Function | Description |
|---|---|
| **NWCCMakeConnPermanent** | Keeps the specified connection from being detached until **NWCCSysCloseConnRef** is called. |
| **NWCCSetPrefServerName** | Sets the *PREFERRED SERVER* |

| | |
|---|---|
| | parameter of the workstation. |
| **NWCCSetPrimConn** | Sets the workstation's primary connection. |

**Parent Topic:**

Connection: Guides

# Connection Table Functions

The following functions operate on the Connection Table for either the NetWare Requester or NETX.

| Function | Description |
|---|---|
| **NWClearConnectionNumber** | Logs out of the specified connection. |
| **NWGetConnectionInformation** | Returns information about a logged in object. |
| **NWGetConnectionUsageStats** | Returns usage statistics for a specified connection. |
| **NWGetConnListFromObject** | Returns a list of connection numbers a specified object has on a server. |
| **NWGetInetAddr** | Returns the network address of the specified connection *connNum* on the specified server. |
| **NWGetObjectConnectionNumbers** | Returns a list of server connection numbers for clients logged in with the specified object name and type. |

**Parent Topic:**

Connection: Guides

# infoType Parameter Values

The *infoType* parameter indicates the type of data desired with one of the following values:

| HEX | Value | Minimum Buffer Size: |
|---|---|---|

| | | Description |
|---|---|---|
| 0x0001 | NWCC_INFO_AUTHENT_STATE | nuint: Returns Authentication state |
| 0x0002 | NWCC_INFO_BCAST_STATE | nuint: Returns Broadcast state |
| 0x0003 | NWCC_INFO_CONN_REF | nuint32: Returns connection reference |
| 0x0004 | NWCC_INFO_TREE_NAME | nstr * length of NW_MAX_TREE_NAME_LEN(33): Returns NDS tree name |
| 0x0005 | NWCC_INFO_CONN_NUMBER | nuint: Returns connection number |
| 0x0006 | NWCC_INFO_USER_ID | nuint32 |
| 0x0007 | NWCC_INFO_SERVER_NAME | nstr * length of NW_MAX_SERVER_NAME_LEN |
| 0x0008 | NWCC_INFO_NDS_STATE | nuint |
| 0x0009 | NWCC_INFO_MAX_PACKET_SIZE | nuint |
| 0x000A | NWCC_INFO_LICENSE_STATE | nuint |
| 0x000B | NWCC_INFO_DISTANCE | nuint |
| 0x000C | NWCC_INFO_SERVER_VERSION | sizeof NWCCVersion |
| 0x000D | NWCC_INFO_TRAN_ADDR | sizeof NWCCTranAddr |

# NWCC_INFO_AUTHENT_STATE Values

NWCC_INFO_AUTHENT_STATE can return one of the following values:

| C Value | Pascal Value | Value Name: Description |
|---|---|---|
| 0x0000 | $0000 | NWCC_AUTHENT_STATE_NONE: Not authenticated |
| 0x0001 | $0001 | NWCC_AUTHENT_STATE_BIND: Bindery authentication |
| | | |

| | | |
|---|---|---|
| 0x0002 | $0002 | NWCC_AUTHENT_STATE_NDS: NDS authentication |

# NWCC_INFO_BCAST_STATE Values

NWCC_INFO_BCAST_STATE can return one of the following:

| C Value | Pascal Value | Value Name: Description |
|---|---|---|
| 0x0000 | $0000 | NWCC_BCAST_PERMIT_ALL: Permit all broadcast messages |
| 0x0001 | $0001 | NWCC_BCAST_PERMIT_SYSTEM: Permit all system broadcast messages |
| 0x0002 | $0002 | NWCC_BCAST_PERMIT_NONE: Do not permit any broadcast messages |
| 0x0003 | $0003 | NWCC_BCAST_PERMIT_POLL: Permit polling to see if any broadcast messages are stored on the server |

# NWCC_INFO_NDS_STATE Values

NWCC_INFO_NDS_STATE can return one of the following:

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0000 | $0000 | NWCC_NDS_NOT_CAPABLE: Server does not support NDS |
| 0x0001 | $0001 | NWCC_NDS_CAPABLE: Server supports NDS |

# NWCC_INFO_LICENSE_STATE Values

NWCC_INFO_LICENSE_STATE can return one of the following:

| C Value | Pascal Value | Value Name: Description |
|---|---|---|
| 0x0000 | $0000 | NWCC_NOT_LICENSED: Connection is |

| | | |
|---|---|---|
| | | not licensed |
| 0x0001 | $0001 | NWCC_CONNECTION_LICENSED: Connection is licensed |
| 0x0002 | $0002 | NWCC_HANDLE_LICENSED: Connection is scheduled to be licensed once it is authenticated |

# Security Values

The *prefSecurityFlags* and *reqSecurityFlags* parameters can point to the following values which may be ORed together:

0x00000100 NWCC_SECURITY_LEVEL_CHECKSUM
0x00000200 NWCC_SECUR_LEVEL_SIGN_HEADERS
0x00000400 NWCC_SECURITY_LEVEL_SIGN_ALL
0x00000800 NWCC_SECURITY_LEVEL_ENCRYPT

**NOTE:** NWCC_SECUR_LEVEL_SIGN_HEADERS and NWCC_SECURITY_LEVEL_SIGN_ALL are exclusive to each other.

# Connection: Functions

# NWCCCloseConn

Closes the specified connection
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows* 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCCloseConn (
   NWCONN_HANDLE   connHandle);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCCloseConn
   (connHandle : NWCONN_HANDLE
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the connection handle to be closed.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x886C | NWE_RESOURCE_LOCK |
| 0x8872 | NWE_INVALID_OWNER |

### *Remarks*

**NWCCCloseConn** is used to close an open connection handle. Calling **NWCCCloseConn** has the opposite effect as the three open functions: **NWCCOpenConnByName**, **NWCCOpenConnByAddr**, **NWCCOpenConnByPref**, and **NWCCOpenConnByRef**. After the connection handle is closed, the handle may not be used again to access the connection.

Under Windows 95, **NWCCCloseConn** waits approximately 30 seconds to allow the connection to be unlicensed and then closes the connection.

**NWCCCloseConn** clears a clients local connection handle while **NWClearConnectionNumber** clears a connection from a file server connection table.

### *NCP Calls*

To be supplied

### *See Also*

**NWCCOpenConnByName**, **NWCCOpenConnByPref**, **NWCCOpenConnByRef**, **NWClearConnectionNumber**

# NWCCGetAllConnInfo

Returns all information for the specified connection
**NetWare Server:** 2.2 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetAllConnInfo (
   NWCONN_HANDLE   connHandle,
   nuint           connInfoVersion,
   pNWCCConnInfo   connInfoBuffer);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetAllConnInfo
  (connHandle : NWCONN_HANDLE;
   connInfoVersion : nuint;
   connInfoBuffer : pNWCCConnInfo
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the connection handle for which to return information.

*connInfoVersion*

(IN) Specifies the connection information version (NWCC_INFO_VERSION_1 or higher).

*connInfoBuffer*

(OUT) Points to the NWCCConnInfo structure containing the returned information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| | |

| | |
|---|---|
| 0x8801 | NWE_CONN_INVALID |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x8869 | NWE_ACCESS_VIOLATION |

## Remarks

The *connInfoVersion* parameter specifies which version of the NWCCConnInfo structure will be used.

If *connInfoVersion* is set to NWCC_INFO_VERSION_2 or higher, the *tranAddr* field of the NWCCConnInfo structure must be set to NULL or initialized.

You must allocate the *connInfoBuffer* parameter. It will be returned with all the connection information.

## NCP Calls

To be supplied

## See Also

**NWGetConnectionInformation**, **NWGetUserInfo**

# NWCCGetAllConnRefInfo

Returns all information for a specified connection reference
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetAllConnRefInfo (
   nuint32          connRef,
   nuint            connInfoVersion,
   pNWCCConnInfo    connInfoBuffer);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetAllConnRefInfo
  (connRef : nuint32;
   connInfoVersion : nuint;
   connInfoBuffer : pNWCCConnInfo
) : NWRCODE;
```

## Parameters

*connRef*

(IN) Specifies the connection reference for which information is to be returned.

*connInfoVersion*

(IN) Specifies the connection information version (NWCC_INFO_VERSION_1 or higher).

*connInfoBuffer*

(OUT) Points to the NWCCConnInfo structure containing the returned information.

## Return Values

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

| | |
|---|---|
| 0x0 | SUCCESS |

| 000 | |
|---|---|
| 0x8801 | NWE_CONN_INVALID |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x8869 | NWE_ACCESS_VIOLATION |

## Remarks

The *connInfoVersion* parameter specifies which version of the NWCCConnInfo structure will be used.

If the *connInfoVersion* parameter is set to NWCC_INFO_VERSION_2 or higher, the *tranAddr* field of the NWCCConnInfo structure must be set to NULL or initialized.

You must allocate the *connInfoBuffer* parameter. It will be returned with all the connection information.

## NCP Calls

To be supplied

## See Also

**NWGetConnectionInformation**, **NWGetUserInfo**

# NWCCGetCLXVersion

Returns the version of the current CLX layer
**NetWare Server:** 2.2, 3.11, 3.12, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

### Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWCCGetCLXVersion (
   pnuint8   majorVersion,
   pnuint8   minorVersion,
   pnuint8   revisionLevel,
   pnuint8   betaReleaseLevel);
```

### Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetCLXVersion
  (majorVersion : nuint8;
   minorVersion : nuint8;
   revisionLevel : nuint8;
   betaReleaseLevel : nuint8
);
```

### Parameters

*majorVersion*

(OUT) Points to the current major version number.

*minorVersion*

(OUT) Points to the current minor version number.

*revisionLevel*

(OUT) Points to the current revision level.

*betaReleaseLevel*

(OUT) Points to the current beta release level.

### Return Values

None

### NCP Calls

*Connection Service Group*

To be supplied

# NWCCGetConnAddress

Returns the transport address for the specified connection
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetConnAddress (
   NWCONN_HANDLE   connHandle,
   nuint32         bufferLen,
   pNWCCTranAddr   tranAddr);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetConnAddress
  (connHandle : NWCONN_HANDLE;
   bufferLen : nuint32;
   tranAddr : pNWCCTranAddr
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the connection handle for which to return the transport address.

*bufferLen*

(IN) Specifies the size (in bytes) of the *buffer* field in the NWCCTranAddr structure.

*tranAddr*

(OUT) Points to the NWCCTranAddr structure containing the transport address.

## Return Values

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

| | |
|---|---|
| 0x0 | SUCCESS |

| 000 | |
|---|---|
| 0x8801 | NWE_CONN_INVALID |
| 0x8867 | NWE_INSUFFICIENT_RESOURCES |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x8869 | NWE_ACCESS_VIOLATION |

## Remarks

Be sure that the *bufferLen* parameter is large enough to contain the returned address. Otherwise, **NWCCGetConnAddress** will fail and return NWE_INSUFFICIENT_RESOURCES. Call the **NWCCGetConnAddressLength** function to determine the address length and then allocate enough memory in the buffer for the *bufferLen* parameter.

The *len* field in the NWCCTranAddr structure will contain the address length upon return.

## NCP Calls

To be supplied

# NWCCGetConnAddressLength

Returns the length of the connection address for the specified connection
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetConnAddressLength (
   NWCONN_HANDLE   connHandle,
   pnuint32        addrLen);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetConnAddressLength
  (connRef : nuint32;
   addrLen : pnuint32
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the connection for which to return the connection address length.

*addrLen*

(OUT) Points to the length of the connection address.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x8 | NWE_ACCESS_VIOLATION |

| 869 | |
|---|---|

## Remarks

**NWCCGetConnAddressLength** returns the length of the connection address in bytes. The *addrLen* parameter should be used to allocate a buffer to pass into the **NWGetConnAddress** or **NWGetConnRefAddress** function.

## NCP Calls

To be supplied

# NWCCGetConnInfo

Returns information about the specified connection
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetConnInfo (
   NWCONN_HANDLE   connHandle,
   nuint           infoType,
   nuint           len,
   nptr            buffer);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetConnInfo
  (connHandle : NWCONN_HANDLE;
   infoType : nuint;
   len : nuint;
   buffer : nptr
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the connection handle for which to return information.

*infoType*

(IN) Specifies the information to be returned about the connection specified in the *connHandle* parameter (length of NW_MAX_TREE_NAME_LEN).

*len*

(IN) Specifies the length of the information buffer to be returned.

*buffer*

(OUT) Points to a buffer containing the returned information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x886B | NWE_INVALID_LEVEL |

## Remarks

**NWCCGetConnInfo** returns a single piece of connection information for the specified connection. It is important that the size of the *buffer* parameter is large enough to contain the requested information.

If the *infoType* parameter is invalid, NWE_INVALID_LEVEL will be returned. If the *infoType* parameter is set to NWCC_INFO_TRAN_ADDR, **NWCCGetConnInfo** will use the NWCCTranAddr structure. See infoType Parameter Values.

See NWCC_INFO_AUTHENT_STATE Values.

See NWCC_INFO_BCAST_STATE Values.

See NWCC_INFO_NDS_STATE Values.

See NWCC_INFO_LICENSE_STATE Values.

## NCP Calls

To be supplied

# NWCCGetConnRef

Returns the connection reference for the specified connection
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetConnRef (
   NWCONN_HANDLE   connHandle,
   pnuint32        connRef);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetConnRef
  (connHandle : NWCONN_HANDLE;
   connRef : pnuint32
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the connection handle for which to return the reference.

*connRef*

(OUT) Points to the connection reference associated with the connection specified by *connHandle*.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |

## NCP Calls

To be supplied

# NWCCGetConnRefAddress

Returns the transport address for the specified connection reference
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetConnRefAddress (
   nuint32          connRef,
   nuint32          bufferLen,
   pNWCCTranAddr    tranAddr);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetConnRefAddress
  (connRef : nuint32;
   bufferLen : nuint32;
   tranAddr : pNWCCTranAddr
) : NWRCODE;
```

## Parameters

*connRef*

(IN) Specifies the connection reference for which to return the transport address.

*bufferLen*

(IN) Specifies the size, in bytes, of the structure field *buffer* of NWCCTranAddr.

*tranAddr*

(OUT) Points to the NWCCTranAddr structure containing the address.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| | |

| | |
|---|---|
| 0x8801 | NWE_CONN_INVALID |
| 0x8867 | NWE_INSUFFICIENT_RESOURCES |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x886C | NWE_RESOURCE_LOCK |
| 0x8872 | NWE_INVALID_OWNER |

## Remarks

You need to ensure that *bufferLen* is large enough to contain the returned address; otherwise, **NWCCGetConnAddress** will fail and return NWE_INSUFFICIENT_RESOURCES. Call **NWCCGetConnAddressLength** to determine the address length and then allocate enough memory in the buffer for *bufferLen*.

*len* in NWCCTranAddr will contain the address length upon return.

## NCP Calls

To be supplied

# NWCCGetConnRefAddressLength

Returns the length of the connection address for the specified connection reference

**NetWare Server:** 2.2, 3.x, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetConnRefAddressLength (
   nuint32    connRef,
   pnuint32   addrLen);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetConnRefAddressLength
  (connRef : nuint32;
   addrLen : nuint32
) : NWRCODE;
```

## Parameters

*connRef*

(IN) Specifies the connection reference for which to return the connection address length.

*addrLen*

(OUT) Points to the length of the connection address.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x880E | NWE_BUFFER_OVERFLOW |
| | |

| | |
|---|---|
| 0x8864 | NWE_INVALID_MATCH_DATA |
| 0x8865 | NWE_MATCH_FAILED |
| 0x8866 | NWE_NO_MORE_ENTRIES |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x886B | NWE_INVALID_LEVEL |
| 0x886C | NWE_RESOURCE_LOCK |

## Remarks

**NWCCGetConnRefAddressLength** returns the length of the connection address in bytes. The *addrLen* parameter should be used to allocate a buffer to pass into the **NWGetConnAddress** or **NWGetConnRefAddress** functions.

If the *infoType* parameter is invalid, NWE_INVALID_LEVEL will be returned. See infoType Parameter Values.

## NCP Calls

To be supplied

# NWCCGetConnRefInfo

Returns the specified information for a given connection reference
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetConnRefInfo (
   nuint32   connRef,
   nuint     infoType,
   nuint     len,
   nptr      buffer);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetConnRefInfo
  (connRef : nuint32;
   infoType : nuint;
   len : nuint;
   buffer : nptr
) : NWRCODE;
```

## Parameters

*connRef*

(IN) Specifies the connection reference for which to return the specified information.

*infoType*

(IN) Specifies the information to be returned about the connection.

*len*

(IN) Specifies the length of the information buffer to be returned.

*buffer*

(OUT) Points to a buffer containing the returned information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x880E | NWE_BUFFER_OVERFLOW |
| 0x8864 | NWE_INVALID_MATCH_DATA |
| 0x8865 | NWE_MATCH_FAILED |
| 0x8866 | NWE_NO_MORE_ENTRIES |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x886B | NWE_INVALID_LEVEL |
| 0x886C | NWE_RESOURCE_LOCK |

## Remarks

**NWCCGetConnRefInfo** returns connection information from the NWCCConnInfo structure associated with the given connection. **NWCCGetConnRefInfo** can either be set to return one field of the structure or the entire structure itself.

*buffer* must point to a buffer of the type of information being requested. (The return type is noted below for cache information.)

If the *infoType* parameter is invalid, NWE_INVALID_LEVEL will be returned. See infoType Parameter Values.

If the *infoType* parameter is set to NWCC_INFO_TRAN_ADDR, **NWCCGetConnRefInfo** will use the NWCCTranAddr structure.

See NWCC_INFO_AUTHENT_STATE Values.

See NWCC_INFO_BCAST_STATE Values.

See NWCC_INFO_NDS_STATE Values.

See NWCC_INFO_LICENSE_STATE Values.

## NCP Calls

*Connection Service Group*

To be supplied

# NWCCGetNumConns

Returns the current and maximum number of connections for the requester
**NetWare Server:** 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetNumConns (
   pnuint     maxConns,
   pnuint     publicConns,
   pnuint     myPrivateConns);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetNumConns
(  Var maxConns : nuint;
   Var publicConns : nuint;
   Var myPrivateConns : nuint
) : NWRCODE;
```

## Parameters

*maxConns*

(OUT) Points to the maximum number of connections allowed with the requester (-1 for requesters with dynamic connection tables).

*publicConns*

(OUT) Points to the current number of public connections (optional).

*myPrivateConns*

(OUT) Points to the current number of private connections owned by the calling process (optional).

See Listing connection numbers: Example.

## Return Values

See Return Values.

## NCP Calls

None

*Connection Service Group*

None

# NWCCGetPrefServerName

Returns the name from the PREFERRED SERVER parameter
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetPrefServerName (
   nuint   len,
   pnstr   prefServer);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetPrefServerName
  (len : nuint;
   prefServer : pnstr
) : NWRCODE;
```

## Parameters

*len*

(IN) Specifies the length of the preferred server string.

*prefServer*

(OUT) Points to a string containing the preferred server name.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |

## Remarks

The *prefServer* parameter is read from the NET.CFG file and is used by the requester to determine which server to attempt to connect to when no

other connections are established, such as during the load process of the requester.

### *NCP Calls*

To be supplied

# NWCCGetPrimConnRef

Returns the primary connection reference
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetPrimConnRef (
   pnuint32   connRef);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCGetPrimConnRef
  (connRef : pnuint32
) : NWRCODE;
```

## Parameters

*connRef*

(OUT) Points to the primary connection reference of the workstation.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |

## Remarks

The primary connection identifies the server to which the user originally logged in. For NDS, the primary connection reference is the connection with the writeable replica used during the login process.

## NCP Calls

To be supplied

# NWCCGetSecurityFlags

Returns the configured security flags for the requester
**NetWare Server:** 4.1x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetSecurityFlags (
   pnuint32   enabSecurityFlags,
   pnuint32   prefSecurityFlags,
   pnuint32   reqSecurityFlags);
```

## Pascal Syntax

```
Function NWCCGetSecurityFlags (
   enabSecurityFlags : pnuint32;
   prefSecurityFlags : pnuint32;
   reqSecurityFlags : pnuint32
) : NWRCODE;
```

## Parameters

*enabSecurityFlags*

(OUT) Points to the security flags which are enabled and supported by the requester and specifies the maximum level the requester can support.

*prefSecurityFlags*

(OUT) Points to the preferred (but not required) security level.

*reqSecurityFlags*

(OUT) Points to the required security flags for each connection.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |

## Remarks

The *enabSecurityFlags*, *prefSecurityFlags*, and *reqSecurityFlags* parameters can point to the following values which may be ORed together:

0x00000100 NWCC_SECURITY_LEVEL_CHECKSUM
0x00000200 NWCC_SECUR_LEVEL_SIGN_HEADERS
0x00000400 NWCC_SECURITY_LEVEL_SIGN_ALL
0x00000800 NWCC_SECURITY_LEVEL_ENCRYPT

**NOTE:** NWCC_SECUR_LEVEL_SIGN_HEADERS and NWCC_SECURITY_LEVEL_SIGN_ALL are exclusive to each other.

If a bit is cleared in the *enabSecurityFlags* parameter bit mask, that same bit cannot be set in either the *prefSecurityFlags* or *reqSecurityFlags* parameter bit masks.

The requester will attempt to establish the level of security defined in the *prefSecurityFlags* parameter on each established connection. However, a connection will not fail if the preferred security level is not supported.

If a server does not support the level of security specified by the *reqSecurityFlags* parameter, the authentication of the connection is not allowed by the requester.

## NCP Calls

None

## See Also

**NWCCSetSecurityFlags**

# NWCCLicenseConn

Licenses the specified connection
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCLicenseConn (
   NWCONN_HANDLE   connHandle);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCLicenseConn
  (connHandle : NWCONN_HANDLE
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies an open connection handle in an unlicensed state.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x8872 | NWE_INVALID_OWNER |

## Remarks

**NWCCLicenseConn** causes a connection to become licensed. If necessary, the license NCP will be sent. If the specified handle is already in a licensed state, an error (NWE_HANDLE_ALREADY_LICENSED) will be returned on most platforms.

Under Windows NT, SUCCESS will be returned if the specified handle is already licensed.

Windows 95 will return SUCCESS as the requester does not support **NWCCLicenseConn**.

**NWCCLicenseConn** is supported under VLMs.

If no connection exists, **NWCCLicenseConn** sets a flag indicating the desire for the connection to be licensed once it has become authenticated.

**NWCCLicenseConn** is not supported on client32 requesters.

See Licensing / Unlicensing a connection: Example.

### *NCP Calls*

To be supplied

# NWCCMakeConnPermanent

Keeps the specified connection from being detached until **NWCCSysCloseConnRef** is called

**NetWare Server:** 2.2, 3.x, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCMakeConnPermanent (
   NWCONN_HANDLE   connHandle);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCMakeConnPermanent
  (connHandle : NWCONN_HANDLE
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the open connection handle associated with the connection to be made permanent.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x8872 | NWE_INVALID_OWNER |

### *Remarks*

**NWCCMakeConnPermanent** keeps the connection from becoming detached until the **NWCCSysCloseConnRef** function is called and allows the connection to remain intact after the termination of all processes having that connection open.

### *NCP Calls*

To be supplied

# NWCCOpenConnByAddr

Opens a connection using a network address
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCOpenConnByAddr (
   pNWCCTranAddr     tranAddr,
   nuint             openState,
   nuint             reserved,
   pNWCONN_HANDLE    pConnHandle);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCOpenConnByAddr
  (tranAddr : pNWCCTranAddr;
   openState : nuint;
   reserved : nuint;
   pConnHandle : pNWCONN_HANDLE
) : NWRCODE;
```

## Parameters

*tranAddr*

(IN) Points to the NWCCTranAddr structure containing the transport address to open the connection to.

*openState*

(IN) Specifies the state of the connection.

*reserved*

(IN) Reserved for future use. Set to NWCC_RESERVED.

*pConnHandle*

(OUT) Points to the connection handle.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8841 | NWE_TRAN_INVALID_TYPE |
| 0x8867 | NWE_INSUFFICIENT_RESOURCES |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x886C | NWE_RESOURCE_LOCK |
| 0x8870 | NWE_UNSUPPORTED_TRAN_TYPE |

### Remarks

The *openState* parameter can have the following values:

| **C Value** | **Pascal Value** | **Value Name** |
|---|---|---|
| 0x0001 | $0001 | NWCC_OPEN_LICENSED |
| 0x0002 | $0002 | NWCC_OPEN_UNLICENSED |

### NCP Calls

To be supplied

# NWCCOpenConnByName

Resolves the given server name to a network address then creates a connection to that address

**NetWare Server:** 2.2, 3.x, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCOpenConnByName (
   NWCONN_HANDLE    startConnHandle,
   pnstr8           name,
   nuint            nameFormat,
   nuint            openState,
   nuint            tranType,
   pNWCONN_HANDLE   pConnHandle);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCOpenConnByName (
   startConnHandle : NWCONN_HANDLE;
   name : pnstr8;
   nameFormat : nuint;
   openState : nuint;
   tranType : nuint;
   Var connHandle : NWCONN_HANDLE
) : NWRCODE;
```

## Parameters

*startConnHandle*

(IN) Specifies the connection to use when resolving the name.

*name*

(IN) Points to the name of the server to which to connect.

*nameFormat*

(IN) Specifies the format of the server name.

*openState*

(IN) Specifies the desired open state of the connection.

*tranType*

(IN) Specifies the transport type.

(IN) Specifies the transport type.

*pConnHandle*

(OUT) Points to the connection handle to be returned and may be used for all requests directed to the connection.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | NWE_CONN_INVALID |
| 0x8808 | NWE_SERVER_NO_SLOTS |
| 0x880A | NWE_SERVER_NO_ROUTE |
| 0x883F | NWE_CONN_TABLE_FULL |
| 0x8841 | NWE_TRAN_INVALID_TYPE |
| 0x8847 | NWE_SERVER_NOT_FOUND |
| 0x8867 | NWE_INSUFFICIENT_RESOURCES |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x8870 | NWE_UNSUPPORTED_TRAN_TYPE |

## Remarks

*startConnHandle* is the connection to use when resolving a name. For instance, if the name is a bindery name, the requester will scan the bindery of the given connection for the required server name.

*startConnHandle* can also be zero if you don't care which connection is used to resolve the name.

*name* points to the structure containing the name of the file server to which to connect. The format and length of these strings are defined by

*nameFormat*.

*tranType* can have the following values:

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0001 | $0001 | NWCC_TRAN_TYPE_IPX |
| 0x0002 | $0002 | NWCC_TRAN_TYPE_UDP |
| 0x0003 | $0003 | NWCC_TRAN_TYPE_DDP |
| 0x0004 | $0004 | NWCC_TRAN_TYPE_ASP |
| 0x8000 | $8000 | NWCC_TRAN_TYPE_WILD |

**NOTE:** Under NETX and VLM, *tranType* can only be set to either NWCC_TRAN_TYPE_IPX or NWCC_TRAN_TYPE_WILD. Otherwise, **NWCCOpenConnByName** will return NWE_UNSUPPORTED_TRAN_TYPE.

*openState* can have the following values:

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0001 | $0001 | NWCC_OPEN_LICENSED |
| 0x0002 | $0002 | NWCC_OPEN_UNLICENSED |

*nameFormat* can have the following values:

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0002 | $0002 | NWCC_NAME_FORMAT_BIND |
| | | |

| | | |
|---|---|---|
| 0x0008 | $00 08 | NWCC_NAME_FORMAT_NDS_TREE |

See Opening a Connection Using a Server Name: Example.

### NCP Calls

To be supplied

### See Also

**NWCCCloseConn, NWCCOpenConnByPref, NWCCOpenConnByRef**

# NWCCOpenConnByPref

Opens an initial connection using the configured preferred settings
**NetWare Server:** 2.2, 3.11, 3.12, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCOpenConnByPref (
   nuint            tranType,
   nuint            openState,
   nuint            reserved,
   pNWCONN_HANDLE   pConnHandle);
```

## Parameters

*tranType*

(IN) Specifies the preferred or required transport type to be used.

*openState*

(IN) Specifies the state of the connection.

*reserved*

(IN/OUT) Reserved for future use. Set to NWCC_RESERVED.

*pConnHandle*

(OUT) Points to the connection handle to be returned.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | NWE_CONN_INVALID |
| 0x8808 | NWE_SERVER_NO_SLOTS |
| 0x880A | NWE_SERVER_NO_ROUTE |
| 0x8 | NWE_CONN_TABLE_FULL |

| | |
|---|---|
| 83F | |
| 0x8847 | NWE_SERVER_NOT_FOUND |
| 0x8867 | NWE_INSUFFICIENT_RESOURCES |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x8870 | NWE_UNSUPPORTED_TRAN_TYPE |

## Remarks

**NWCCOpenConnByPref** is similar to **NWCCOpenConnByName**, which uses the preferred server or preferred tree name, except that **NWCCOpenConnByPref** uses the configured preferences of the requester to establish an initial connection to a server.

**NOTE:** In the event that a connection to the preferred tree or server cannot be established, another connection may be returned.

**NWCCOpenConnByPref** will return NWE_CONN_INVALID if the platform being run is not Windows 95 since **NWCCOpenConnByPref** is only successful on Windows 95.

The *tranType* parameter must be set to indicate the transport type desired with the following values:

0x0001 NWCC_TRAN_TYPE_IPX
0x0002 NWCC_TRAN_TYPE_UDP
0x0003 NWCC_TRAN_TYPE_DDP
0x0004 NWCC_TRAN_TYPE_ASP

The *openState* parameter must be set to indicate the type of data desired with the following values:

0x0001 NWCC_OPEN_LICENSED
0x0002 NWCC_OPEN_UNLICENSED

## NCP Calls

To be supplied

## See Also

**NWCCCloseConn**, **NWCCOpenConnByName**, **NWCCOpenConnByRef**

# NWCCOpenConnByRef

Opens a connection associated with the given connection reference
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCOpenConnByRef (
   nuint32           connRef,
   nuint             openState,
   nuint             reserved,
   pNWCONN_HANDLE     pConnHandle);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCOpenConnByRef
  (connRef : nuint32;
   openState : nuint;
   reserved : nuint;
   pConnHandle : pNWCONN_HANDLE
) : NWRCODE;
```

## Parameters

*connRef*

(IN) Specifies a reference, which identifies a valid connection.

*openState*

(IN) Specifies the state of the connection.

*reserved*

(IN) Reserved for future use. Set to NWCC_RESERVED.

*pConnHandle*

(OUT) Points to the connection handle to be returned.

## Return Values

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

| | |
|---|---|

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x8836 | INVALID_PARAMETER |
| 0x8869 | NWE_ACCESS_VIOLATION |

## Remarks

**NWCCScanConnRefs** can be called to get the connection reference.

*connRef* can be used to get information about the connection, but a valid connection handle must be used to make actual requests to the connection.

*openState* can have the following values:

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0001 | $0001 | NWCC_OPEN_LICENSED |
| 0x0002 | $0002 | NWCC_OPEN_UNLICENSED |

## NCP Calls

To be supplied

## See Also

**NWCCCloseConn**, **NWCCOpenConnByName**, **NWCCOpenConnByPref**

# NWCCQueryFeature

Determines if the Requester supports a given feature
**NetWare Server:** 2.2, 3.11, 3.12, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCQueryFeature (
   nuint   featureCode);
```

## Parameters

*featureCode*

(IN) Specifies the feature being queried.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x88FF | NWE_REQUESTER_FAILURE |

## Remarks

**NWCCQueryFeature** allows requesters to add incremental support for the NWClient interface without requiring the larger libraries like NWCalls and NWNet to keep track of requester versions and whether or not they support a specific feature.

*featureCode* must be set to indicate the type of data desired with the following values:

0x0001 NWCC_FEAT_PRIV_CONN
0x0002 NWCC_FEAT_REQ_AUTH
0x0003 NWCC_FEAT_SECURITY
0x0004 NWCC_FEAT_NDS
0x0005 NWCC_FEAT_NDS_MTREE

0x0006 NWCC_FEAT_PRN_CAPTURE

### *NCP Calls*

To be supplied

# NWCCRenegotiateSecurityLevel

Sets a new security level for the specified connection
**NetWare Server:** 4.1x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCRenegotiateSecurityLevel (
   NWCONN_HANDLE   connHandle,
   nuint32         securityFlags);
```

## Pascal Syntax

```
Function NWCCRenegotiateSecurityLevel (
   connHandle : NWCONN_HANDLE;
   securityFlags : nuint32
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the connection handle.

*securityFlags*

(IN) Specifies the desired level of security for the specified connection.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x8836 | NWE_PARAM_INVALID |
| 0x8861 | NWE_SIGNATURE_LEVEL_CONFLICT |
| 0x8869 | NWE_ACCESS_VIOLATION |

## Remarks

In order to establish a new level of security with a server, the requester must compare the desired security level with its own supported level and the support level of the server. The actual security level cannot exceed the supported levels of either the requester or the server.

Any changes in security levels will not actually occur until the next authentication on the connection.

Before reducing the security level, you must first close the connection by calling the **NWCCSysCloseConnRef** function and then reopen the connection by calling one of the NWCCOpenConnBy* functions.

The *securityFlags* parameter can point to the following values which may be ORed together:

| | |
|---|---|
| 0x00000000 | NWCC_SECUR_SIGNING_NOT_IN_USE |
| 0x00000001 | NWCC_SECURITY_SIGNING_IN_USE |
| 0x00000100 | NWCC_SECURITY_LEVEL_CHECKSUM |
| 0x00000200 | NWCC_SECUR_LEVEL_SIGN_HEADERS |
| 0x00000400 | NWCC_SECURITY_LEVEL_SIGN_ALL |
| 0x00000800 | NWCC_SECURITY_LEVEL_ENCRYPT |

**NOTE:** NWCC_SECUR_SIGNING_NOT_IN_USE and NWCC_SECURITY_SIGNING_IN_USE are exclusive to each other.

NWCC_SECUR_LEVEL_SIGN_HEADERS and NWCC_SECURITY_LEVEL_SIGN_ALL are exclusive to each other.

## NCP Calls

None

## See Also

**NWCCOpenConnByAddr**, **NWCCOpenConnByName**, **NWCCOpenConnByPref**, **NWCCOpenConnByRef**, **NWCCSysCloseConnRef**

# NWCCScanConnInfo

Returns connection information for multiple connections
**NetWare Server:** 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCScanConnInfo (
   pnuint32   scanIterator,
   nuint      scanInfoLevel,
   nptr       scanConnInfo,
   nuint      scanFlags,
   nuint      connInfoVersion,
   nuint      returnInfoLevel,
   pnuint     returnConnInfo,
   pnuint32   connReference);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCScanConnInfo
(  scanIterator : pnuint32;
   scanInfoLevel : nuint;
   scanConnInfo : nptr;
   scanFlags : nuint;
   connInfoVersion : nuint;
   returnInfoLevel : nuint;
   returnConnInfo : nptr;
   Var connReference : nuint32
) : NWRCODE;
```

## Parameters

*scanIterator*

(IN/OUT) Inputs the iterator handle used. Must be initialized to zero (0) for first scan and to restart a search. Outputs the next iteration number; do not alter on subsequent scans.

*scanInfoLevel*

(IN) Specifies the data type of the *scanConnInfo* parameter.

*scanConnInfo*

(IN) Points to the search data used during the scan.

*scanFlags*

(IN) Specifies which type of connections (licensed/unlicensed and public/private) to search and if the data passed into the *scanConnInfo* parameter needs to match prospective connections.

*connInfoVersion*

(IN) Specifies the connection information version. Set to NWCC_INFO_VERSION_1 or higher.

*returnInfoLevel*

(IN) Specifies the data type of the *returnConnInfo* parameter.

*returnConnInfo*

(OUT) Points to the returned information and is of the data type specified in the *returnInfoLevel* parameter.

*connReference*

(OUT) Points to the connection reference associated with the returned information (optional).

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x880E | NWE_BUFFER_OVERFLOW |
| 0x8864 | NWE_INVALID_MATCH_DATA |
| 0x8866 | NWE_NO_MORE_ENTRIES |
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x886B | NWE_INVALID_LEVEL |

## Remarks

The *scanInfoLevel* and *returnInfoLevel* parameters can have the following values (which correspond to the fields found in the NWCCConnInfo structure):

| **Val** | **Value Name** | **Data Type of the *returnConnInfo*** |
|---|---|---|

| ue | | parameter |
|---|---|---|
| 0x0000 | NWCC_INFO_NONE | NULL |
| 0x0001 | NWCC_INFO_AUTHENT_STATE | pnuint |
| 0x0002 | NWCC_INFO_BCAST_STATE | pnuint |
| 0x0003 | NWCC_INFO_CONN_REF | pnuint32 |
| 0x0004 | NWCC_INFO_TREE_NAME | pnstr of length NW_MAX_TREE_NAME_LEN |
| 0x0005 | NWCC_INFO_CONN_NUMBER | pnuint |
| 0x0006 | NWCC_INFO_USER_ID | pnuint32 |
| 0x0007 | NWCC_INFO_SERVER_NAME | pnstr of length NW_MAX_SERVER_NAME_LEN |
| 0x0008 | NWCC_INFO_NDS_STATE | pnuint |
| 0x0009 | NWCC_INFO_MAX_PACKET_SIZE | pnuint |
| 0x00A | NWCC_INFO_LICENSE_STATE | pnuint |
| 0x00B | NWCC_INFO_DISTANCE | pnuint |
| 0x00C | NWCC_INFO_SERVER_VERSION | pNWCCVersion |
| 0x00D | NWCC_INFO_TRAN_ADDR | pNWCCTranAddr |

The *returnInfoLevel* parameter can also contain the 0xFFFF NWCC_INFO_RETURN_ALL value of type pNWCCConnInfo.

Either the entire NWCCConnInfo structure or part of the NWCCConnInfo structure can be returned using the *returnConnInfo* parameter. If the entire NWCCConnInfo structure is being returned ( *returnInfoLevel*=NWCC_INFO_RETURN_ALL), the *returnConnInfo* parameter must point to a buffer of type NWCCConnInfo. If part of the NWCCConnInfo structure is being returned, the *returnConnInfo* parameter must point to a buffer of the data type being requested.

If the return value for **NWCCScanConnInfo** is BUFFER_OVERFLOW when using the NWCC_INFO_TRAN_ADDR or NWCC_INFO_RETURN_ALL value for the *returnInfoLevel* parameter, the *len* field in the NWCCTranAddr structure was passed an incorrect

amount. The **NWCCGetConnRefInfo** function can be called to retrieve the transport address.

*scanFlags* can have the following values:

| | |
|---|---|
| 0x0 000 | NWCC_MATCH_NOT_EQUALS: Specifies every connection not matching a given data member should be scanned. |
| 0x0 001 | NWCC_MATCH_EQUALS: Specifies every connection matching a given data member should be scanned. |
| 0x0 002 | NWCC_RETURN_PUBLIC: Specifies all public connections should be considered in the scan. |
| 0x0 004 | NWCC_RETURN_PRIVATE: Specifies all private connections should be considered in the scan. |
| 0x0 008 | NWCC_RETURN_LICENSED: Specifies that only licensed connections are desired. |
| 0x0 010 | NWCC_RETURN_UNLICENSED: Specifies that only unlicensed connections are desired. |

**NOTE:** Note that NWCC_MATCH_NOT_EQUALS and NWCC_MATCH_EQUALS may not be set simultaneously while all of the other values for the *scanFlags* parameter may be ORed together. Also, when the *scanInfoLevel* parameter is set to NWCC_INFO_NONE, NWCC_MATCH_NOT_EQUALS and NWCC_MATCH_EQUALS are ignored.

To have **all** public and private connections considered in a scan, do one of these two:

Use both NWCC_RETURN_PUBLIC and NWCC_RETURN_PRIVATE.

Do not use either NWCC_RETURN_PUBLIC or NWCC_RETURN_PRIVATE.

To have **all** licensed and unlicensed connections considered in a scan, do one of these two:

Use both NWCC_RETURN_LICENSED and NWCC_RETURN_UNLICENSED.

Do not use either NWCC_RETURN_LICENSED or NWCC_RETURN_UNLICENSED.

### NCP Calls

To be supplied

# NWCCScanConnRefs

Returns a connection reference for each connection on the workstation
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCScanConnRefs (
   pnuint32   scanIterator,
   pnuint32   connRef);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCScanConnRefs
  (scanIterator : pnuint32;
   connRef : pnuint32
) : NWRCODE;
```

## Parameters

*scanIterator*

(IN/OUT) Points to an iterator (zero on the first scan).

*connRef*

(OUT) Points to the connection reference for each connection.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x880E | NWE_BUFFER_OVERFLOW |
| 0x8864 | NWE_INVALID_MATCH_DATA |
| 0x8866 | NWE_NO_MORE_ENTRIES |
| | |

| | |
|---|---|
| 0x8868 | NWE_STRING_TRANSLATION |
| 0x886B | NWE_INVALID_LEVEL |

### Remarks

The *scanIterator* parameter must not be altered on subsequent scans.

### NCP Calls

To be supplied

# NWCCSetCurrentConnection

Sets the current connection ID and current connection for the thread group control structure

**Local Servers:** nonblocking

**Local Servers:** nonblocking

**NetWare Server:** 4.x

**Platform:** NLM

**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCSetCurrentConnection (
   CONN_HANDLE   connHandle);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCSetCurrentConnection
  (connHandle : CONN_HANDLE
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the connection handle.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x890A | INVALID_CONNECTION |

## Remarks

**NWCCSetCurrentConnection** is similar to the **SetCurrentConnection** and **SetCurrentConnectionID** functions, although

and **SetCurrentConnectionID** functions, although **NWCCSetCurrentConnection** will accept a connection obtained through the CLX library.

If you open a connection by calling the **NWCCOpenConnByName** function, you can call **NWCCSetCurrentConnection** and pass the connection handle returned by the **NWCCOpenConnByName** function. The **fopen** function can then be called by specifying a file on the server for which a connection was recently opened. The connection opened by calling the **NWCCOpenConnByName** function will be used.

You can use the old CLIB connection model by calling **NWCCSetCurrentConnection** followed by calling the **GetCurrentServerID** and **GetCurrentConnection** functions.

**NOTE:** To bridge from a connection opened by an old NIT connection function such as the **AttachToFileServer** function, set your current server ID and pass in the connection allocated by the **AttachToFileServer** function as the connection handle parameter for any NWCalls function.

**NOTE:** If **NWCCSetCurrentConnection** is called from any platform other than NLM, SUCCESS will be returned but no action will be performed.

## NCP Calls

None

## See Also

**fopen**, **NWCCOpenConnByName**, **SetCurrentConnection**, **SetCurrentConnectionID**

# NWCCSetPrefServerName

Sets the *prefServer* parameter for the workstation
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCSetPrefServerName (
   pnstr   prefServer);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCSetPrefServerName
  (prefServer : pnstr
) : NWRCODE;
```

## Parameters

*prefServer*

(IN) Points to the string containing the preferred server name.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |

## Remarks

The *prefServer* parameter is read in from the NET.CFG file and is used by the requester and determines to which server to connect when no other connections are established, such as during the requester load process.

## NCP Calls

To be supplied

# NWCCSetPrimConn

Sets the primary connection for the workstation
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCSetPrimConn (
   NWCONN_HANDLE   connHandle);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCSetPrimConn
  (connHandle : NWCONN_HANDLE
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies the connection handle to make primary.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |

## Remarks

The primary connection identifies the server to which the user originally logged in. For NDS, the primary connection is the connection with the writeable replica used during the login process.

## NCP Calls

To be supplied

# NWCCSetSecurityFlags

Sets the configured security flags for the requester
**NetWare Server:** 4.1x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCSetSecurityFlags (
   nuint32   prefSecurityFlags,
   nuint32   reqSecurityFlags);
```

## Pascal Syntax

```
Functioin NWCCSetSecurityFlags (
   prefSecurityFlags : nuint32;
   reqSecurityFlags : nuint32
) : NWRCODE;
```

## Parameters

*prefSecurityFlags*

(IN) Specifies the preferred (but not required) security level.

*reqSecurityFlags*

(IN) Specifies the required security flags for each connection.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8836 | NWE_PARAM_INVALID |
| 0x8861 | NWE_SIGNATURE_LEVEL_CONFLICT |

## Remarks

See Security Values.

### NCP Calls

None

### See Also

**NWCCGetSecurityFlags**

# NWCCSysCloseConnRef

Closes and detaches the specified connection, including the connection reference and all connection handles for this connection

**NetWare Server:** 2.2, 3.x, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCSysCloseConnRef (
   nuint32   connRef);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCSysCloseConnRef
  (connRef : nuint32
) : NWRCODE;
```

## Parameters

*connRef*

(IN) Specifies the connection handle to be destroyed.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x8869 | NWE_ACCESS_VIOLATION |

## Remarks

**NWCCSysCloseConnRef** is similar to the **NWCCCloseConn** function.

The exception is that **NWCCSysCloseConnRef** forces all of the open handles to the connection to be closed and detaches the connection.

**NWCCSysCloseConnRef** is a system level request that causes all processes that are accessing this connection to lose access to the resources on the connection.

### *NCP Calls*

To be supplied

# NWCCUnlicenseConn

Unlicenses the specified licensed connection
**NetWare Server:** 2.2, 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCUnlicenseConn (
   NWCONN_HANDLE   connHandle);
```

## Pascal Syntax

```
#include <nwclxcon.inc>

Function NWCCUnlicenseConn
  (connHandle : NWCONN_HANDLE
) : NWRCODE;
```

## Parameters

*connHandle*

(IN) Specifies an open connection handle to be unlicensed.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x8801 | NWE_CONN_INVALID |
| 0x8815 | NWE_HANDLE_ALREADY_UNLICENSED |
| 0x8869 | NWE_ACCESS_VIOLATION |
| 0x8872 | NWE_INVALID_OWNER |

### *Remarks*

A licensed connection can be unlicensed by calling **NWCCUnlicenseConn**. In the requester, **NWCCUnlicenseConn** will only unlicense the connection if there are no other open handles to that connection that need to remain licensed.

### *NCP Calls*

To be supplied

# NWClearConnectionNumber

Logs out the specified connection
**Local Servers:** blocking
**Remote Servers:** blocking
**NetWare Server:** 2.2, 3.11, 3.12, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

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

NWCCODE N_API NWClearConnectionNumber (
   NWCONN_HANDLE   connHandle,
   NWCONN_NUM      connNumber);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWClearConnectionNumber
  (connHandle : NWCONN_HANDLE;
   connNumber : NWCONN_NUM
) : NWCCODE;
```

## Parameters

*connHandle*

(IN) Specifies the server connection handle.

*connNumber*

(IN) Specifies the connection number to be cleared.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| | |

| | |
|---|---|
| 0x8 9C6 | NO_CONSOLE_PRIVILEGES |
| 0x8 9FD | BAD_STATION_NUMBER |

## Remarks

You must have SUPERVISOR or equivalent rights to call **NWClearConnectionNumber**. Otherwise, NO_CONSOLE_PRIVILEGES will be returned.

**NWClearConnectionNumber** clears a connection from a file server connection table while the **NWCCCloseConn** function clears a local connection handle for a client.

## NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 210 Clear Connection Number
0x2222 23 254 Clear Connection Number (3.11+)

## See Also

**NWCCCloseConn**

# NWCLXInit

Initializes the CLX library
**NetWare Server:** 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY (NWRCODE) NWCLXInit (
   nptr   reserved1,
   nptr   reserved2);
```

## Pascal Syntax

```
Function NWCLXInit  (
   reserved1 : nptr;
   reserved2 : nptr
) : NWCCODE;
```

## Parameters

*reserved1*

(IN) Is reserved (pass in NULL).

*reserved2*

(IN) Is reserved (pass in NULL).

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |

## Remarks

**IMPORTANT:** **NWCLXInit** must be called before calling any other NWCC function.

## NCP Calls

None

## *See Also*

**NWCLXTerm**

# NWCLXTerm

Terminates the CLX library and performs any necessary clean up
**NetWare Server:** 3.x, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY (NWRCODE) NWCLXTerm (
   nptr   reserved);
```

## Pascal Syntax

```
Function NWCLXTerm (
    reserved : nptr
) : NWCCODE;
```

## Parameters

*reserved*

(IN) Is reserved; pass in NULL.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |

## NCP Calls

None

## See Also

**NWCLXInit**

# NWFreeConnectionSlot

Either removes all task dependencies on a task disconnect or completely tears down the connection for a system disconnect

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWFreeConnectionSlot
  (NWCONN_HANDLE   conn,
   nuint8          disconnectType);
```

## Pascal Syntax

```
Function NWFreeConnectionSlot
  (conn : NWCONN_HANDLE;
   disconnectType : nuint8
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the connection handle of the desired connection.

*disconnectType*

(IN) Specifies a system disconnect or a task disconnect.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x8 | INVALID_PARAMETER |

| | |
|---|---|
| 836 | |
| 0xF EA0 | ERR_NO_WRITABLE_REPLICAS |
| 0xF EB8 | ERR_CONTEXT_CREATION |

## Remarks

A task disconnect keeps track of a connection in use count. When the count sets to zero, the connection stays valid until its slot is needed for a new connection.

A system disconnect tears down the connection completely. The task disconnect decrements all the in-use counts to zero.

NETX does not support**NWFreeConnectionSlot** and will return an error if VLMs are not running.

Under Client32, **NWFreeConnectionSlot** will return SUCCESSFUL even if the connection being freed is the monitored connection. **NWFreeConnectionSlot** makes a copy of the connection handle from the Client32 requestor. It is this connection handle copy that will be freed even if it is the monitored connection. However, the original connection handle still exists.

**NWFreeConnectionSlot** will try to find another server to store the attributes. If the server is not one of the connections for the client, a connection will be made to the new server that has a writable replica. This connection will become the monitored connection with the login attributes while the original monitored connection will be freed.

If **NWFreeConnectionSlot** cannot find another writeable replica when called to delete the monitored connection, one of the following two errors will be returned:

```
ERR_CONTEXT_CREATION
ERR_NO_WRITABLE_REPLICAS
```

The *disconnectType* parameter will be one of the following:

```
SYSTEM_DISCONNECT
TASK_DISCONNECT
```

ERR_CONTEXT_CREATION is returned sometimes when the unicode tables have not been initialized.

If ERR_NO_WRITABLE_REPLICAS is returned, the connection cannot be deleted until the **NWDSLogout** function has been called since the server has attributes that were created at login time.

## NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 22 Get Station's Logged Info (old)
0x2222 23 28 Get Station's Logged Info
0x2222 104 01 Ping for NDS NCP
0x2222 104 02 Send NDS Fragmented Request/Reply

### See Also

**NWCCOpenConnByAddr, NWCCScanConnRefs, NWCCOpenConnByRef, NWCCLicenseConn**

# NWGetConnectionHandle (obsolete 6/96)

Returns the workstations connection handle for the specified NetWare server but is now obsolete. Call **NWCCScanConnRefs** followed by calling **NWCCOpenConnByRef** instead.

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetConnectionHandle
   (pnuint8                 serverName,
    nuint16                 reserved1,
    NWCONN_HANDLE N_FAR *   conn,
    pnuint16                reserved2);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetConnectionHandle
   (pServerName : pnuint8;
    reserved1 : nuint16;
    Var pConnHandle : NWCONN_HANDLE;
    reserved2 : pnuint16
) : NWCCODE;
```

## Parameters

*serverName*

(IN) Points to the name of the target NetWare server.

*reserved1*

Is reserved (must be 0).

*conn*

(OUT) Points to the variable for the connection handle.

*reserved2*

Is reserved (pass NULL).

### *Return Values*

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x880F | NO_CONNECTION_TO_SERVER |

### *NCP Calls*

0x2222 23 17 Get File Server Information
0x2222 23 22 Get Station's Logged Info (old)
0x2222 23 28 Get Station's Logged Info
0x2222 104 1 Ping for NDS NCP

# NWGetConnectionIDFromAddress (obsolete 9/97)

Returns the connection handle associated with the given NetWare address of the server but is now obsolete. Call the **NWCCScanConnInfo**, **NWCCOpenConnByRef**, and **NWCCLicenseConn** functions instead.

**NetWare Server:** 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWGetConnectionIDFromAddress
  (nuint8                  transType,
   nuint32                 transLen,
   nuint8 N_FAR *          transBuf,
   NWCONN_HANDLE N_FAR *   conn);
```

## Pascal Syntax

```
Function NWGetConnectionIDFromAddress
  (transType : nuint8;
   transLen : nuint32;
   transBuf : pnuint8;
   Var conn : NWCONN_HANDLE
) : NWCCODE;
```

## Parameters

*transType*

(IN) Specifies the transport protocol for the given transport buffer.

*transLen*

(IN) Specifies the transport buffer length (for example: IPX=12).

*transBuf*

(IN) Points to the transport-dependent address information (12 bytes for IPX).

*conn*

(OUT) Points to the connection handle associated with the given NetWare address.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x880F | NO_CONNECTION_TO_SERVER |
| 0x8841 | BAD_TRAN_TYPE |

## Remarks

**NWGetConnectionIDFromAddress (obsolete 9/97)** will return a licensed connection to the NetWare server if the connection is authenticated. Call the **NWDSUnlockConnection** function to unlicense the connection or place it on the LRU list to be cached.

## NCP Calls

None

## See Also

**NWCCGetConnInfo**, **NWCCLicenseConn**, **NWCCOpenConnByAddr**, **NWCCOpenConnByRef**, **NWCCScanConnInfo**, **NWCCScanConnRefs**, **NWGetConnectionInformation**

# NWGetConnectionIDFromName (obsolete 9/97)

Returns the connection to a specified NetWare server but is now obsolete. Call the **NWCCScanConnInfo**, **NWCCOpenConnByRef**, and **NWCCLicenseConn** functions instead.

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 3.x, 4.x (only with the NetWare DOS Requester)

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWGetConnectionIDFromName (
   nuint32                                    nameLen,
   nuint8 N_FAR                            *name,
   NWCONN_HANDLE N_FAR  *conn);
```

## Pascal Syntax

```
Function NWGetConnectionIDFromName
  (nameLen : nuint32;
   name : pnuint8;
   Var conn : NWCONN_HANDLE
) : NWCCODE;
```

## Parameters

*nameLen*

(IN) Specifies the length of the server name.

*name*

(IN) Points to the server name to look up in the connection table.

*conn*

(OUT) Points to the connection handle to the server name.

## Return Values

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

| | |
|---|---|

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |

## Remarks

**NWGetConnectionIDFromName (obsolete 9/97)** only works with the NetWare DOS Requester and will not work if NETX is being used as the client shell. If NETX is loaded, **NWGetConnectionIDFromName (obsolete 9/97)** returns 0x89FF.

**NWGetConnectionIDFromName (obsolete 9/97)** will return a licensed connection to the NetWare server if the connection is authenticated. Call the **NWDSUnlockConnection** function to unlicense the connection or place it on the LRU list to be cached.

## NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 22 Get Station's Logged Info (old)
0x2222 23 28 Get Station's Logged Info
0x2222 104 01 Ping for NDS NCP

## See Also

**NWCCGetConnInfo**, **NWCCLicenseConn**, **NWCCOpenConnByAddr**, **NWCCOpenConnByRef**, **NWCCScanConnInfo**, **NWCCScanConnRefs**, **NWGetConnectionInformation**

# NWGetConnectionInformation

Returns information about a logged in object
**Local Servers:** blocking
**Remote Servers:** blocking
**NetWare Server:** 2.2, 3.11, 3.12, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetConnectionInformation (
   NWCONN_HANDLE   connHandle,
   NWCONN_NUM      connNumber,
   pnstr8          pobjName,
   pnuint16        pobjType,
   pnuint32        pobjID,
   pnuint8         ploginTime);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetConnectionInformation
  (connHandle : NWCONN_HANDLE;
   connNumber : NWCONN_NUM;
   pObjName : pnstr8;
   pObjType : pnuint16;
   pObjID : pnuint32;
   pLoginTime : pnuint8
) : NWCCODE;
```

## Parameters

*connHandle*

(IN) Specifies the NetWare server connection handle.

*connNumber*

(IN) Specifies the NetWare server connection number for which the information is being obtained.

*pObjName*

(OUT) Points to the name of the object whose connection number is passed in *connNumber* (48 bytes, optional).

*pObjType*

(OUT) Points to the bindery object type of the client (optional).

*pObjID*

(OUT) Points to the bindery object ID of the client (optional).

*pLoginTime*

(OUT) Points to the time value when the object logged in at the specified connection number (7 bytes, optional).

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x8996 | SERVER_OUT_OF_MEMORY |
| 0x89FC | NO_SUCH_OBJECT |
| 0x89FD | BAD_STATION_NUMBER |
| 0x89FE | DIRECTORY_LOCKED |
| 0x89FF | HARDWARE_FAILURE |

## Remarks

The *pObjName*, *pObjType*, *pObjID*, and *pLoginTime* parameter are included in the returned information.

The system time clock is a 7-byte value contained in the *pLoginTime* parameter and defined in the following format:

| **Byte** | **Value** | **Range** |
|---|---|---|
| 1 | Year | 0 through 179 |
| 2 | Month | 1 through 12 |
| | | |

| | | |
|---|---|---|
| 3 | Day | 1 through 31 |
| 4 | Hour | 0 through 23 (0 = 12 midnight; 23 = 11 PM) |
| 5 | Minute | 0 through 59 |
| 6 | Second | 0 through 59 |
| 7 | Day of Week | 0 through 6, 0=Sunday |

**NOTE:** For the year value, 80-99=1980-1999; 100-179=2000-2079. The range 0-79 applies to 1900-1979, but a year in this range should not be necessary since DOS cannot return a year value previous to 1980.

## NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 22 Get Station's Logged Info (old)
0x2222 23 28 Get Station's Logged Info

## See Also

**NWCCGetAllConnInfo**, **NWGetUserInfo**

# NWGetConnectionList (obsolete 6/96)

Returns a list of all connection handles but is now obsolete. Call **NWCCScanConnRefs** followed by calling **NWCCOpenConnByRef** instead.

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetConnectionList
   (nuint16                 Mode,
    NWCONN_HANDLE N_FAR *   connListBuffer,
    nuint16                 connListSize,
    pnuint16                numConns);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetConnectionList
   (Mode : nuint16;
    Var connListBuffer : NWCONN_HANDLE;
    connListSize : nuint16;
    pNumConns : pnuint16
) : NWCCODE;
```

## Parameters

*Mode*

Is reserved (must be 0).

*connListBuffer*

(OUT) Points to the connection list for the workstation.

*connListSize*

(IN) Specifies the number of connections the *connListBuffer* parameter can hold.

*numConns*

(OUT) Points to the number of connections returned in the

*connListBuffer* parameter.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x880D | CONNECT_LIST_OVERFLOW |

## NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 22 Get Station's Logged Info (old)
0x2222 23 28 Get Station's Logged Info
0x2222 104 1 Ping for NDS NCP

# NWGetConnectionNumber (obsolete 6/96)

Returns the connection number the requesting workstation uses to communicate with the NetWare server (corresponds to the connection handle) but is now obsolete. Call **NWCCGetConnInfo** instead.

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetConnectionNumber
   (NWCONN_HANDLE         conn,
    NWCONN_NUM N_FAR *    connNumber);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetConnectionNumber
   (connHandle : NWCONN_HANDLE;
    Var connNumber : NWCONN_NUM
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the connection handle for the NetWare server.

*connNumber*

(OUT) Points to the variable containing the corresponding connection number for the workstation on the NetWare server.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| | |

| | |
|---|---|
| 0x8801 | INVALID CONNECTION |

## Remarks

The NetWare server maintains a connection table including connection handles and passwords for all connected objects. After calling **NWGetConnectionNumber (obsolete 6/96)**, the *connNumber* parameter contains the index into the connection table.

## NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 22 Get Station's Logged Info (old)
0x2222 23 28 Get Station's Logged Info
0x2222 104 1 Ping for NDS NCP

## See Also

**NWCCGetConnInfo, NWGetConnectionInformation**

# NWGetConnectionStatus (obsolete 9/97)

Returns status information (server name, address, object type, etc.) about a specified connection handle but is now obsolete. Call the **NWCCGetConnInfo** and **NWGetConnectionInformation** functions instead.

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetConnectionStatus
   (NWCONN_HANDLE           conn,
    CONNECT_INFO N_FAR *    connInfo,
    nuint16                 connInfoSize);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetConnectionStatus
   (connHandle : NWCONN_HANDLE;
    Var pConnInfo : CONNECT_INFO;
    connInfoSize : nuint16
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the connection handle of the NetWare server.

*connInfo*

(OUT) Points to the CONNECT_INFO structure.

*connInfoSize*

(IN) Specifies the size of the *connStatusBuffe\r* field.

## Return Values

These are common return values; see Return Values for more

information.

| | |
|---|---|
| 0x0<br>000 | SUCCESSFUL |
| 0x8<br>801 | INVALID_CONNECTION |

### Remarks

NETX does not support CONNECTION_BROADCAST_AVAILABLE.

### NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 22 Get Station's Logged Info (old)
0x2222 23 28 Get Station's Logged Info
0x2222 104 1 Ping for NDS NCP

# NWGetConnectionUsageStats

Returns the usage statistics for a specified connection
**Local Servers:** blocking
**Remote Servers:** blocking
**NetWare Server:** 2.2
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetConnectionUsageStats (
   NWCONN_HANDLE     conn,
   NWCONN_NUM        connNumber,
   CONN_USE N_FAR  *statusBuffer);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetConnectionUsageStats
  (connHandle : NWCONN_HANDLE;
   connNumber : NWCONN_NUM;
   Var pStatusBuffer : CONN_USE
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the NetWare server connection handle.

*connNumber*

(IN) Specifies the server connection number.

*statusBuffer*

(OUT) Points to the CONN_USE structure to receive the usage statistics.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x89C6 | NO_CONSOLE_OPERATOR |
| 0x89FD | UNKNOWN_REQUEST |

## Remarks

The client must have console rights to get usage statistics for connections other than a client connection.

## NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 229 Get Connection Usage Statistic.

# NWGetConnInfo (obsolete 6/96)

Retrieves specified information about a connection but is now obsolete. Call **NWCCGetAllConnInfo** or **NWCCGetConnAddress** instead.

**NetWare Server:** PNW, 2.2, 3.11, 3.12, 4.x

**Platform:** DOS (VLM only), NLM, OS/2, Windows 3.1, Windows NT (VLM only), Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetConnInfo
   (NWCONN_HANDLE   conn,
    nuint16         type,
    nptr            data);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetConnInfo
   (connHandle : NWCONN_HANDLE;
    connType : nuint16;
    pData : nptr
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the connection for which to return information.

*type*

(IN) Specifies the type of information to retrieve.

*data*

(OUT) Points to a return buffer containing the desired information (size depends on the type of information desired: usually an nuint16, nuint32 or a string).

## Return Values

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

| | |
|---|---|
| 0x8801 | INVALID_CONNECTION |
| 0x8811 | INVALID_SHELL_CALL |
| 0x8836 | INVALID_PARAMETER |

## Remarks

The *type* parameter can have the following values:

3 NW_AUTHENTICATED: Returns 1 if authenticated. Otherwise an nuint16.

13 NW_CONN_NUM: Returns a nuint16.

0x8001 NW_SERVER_ADDRESS: Returns a 12 byte address.

0x8002 NW_SERVER_NAME: Returns a 48 byte string containing the server name.

INVALID_PARAMETER will be returned if the *type* parameter is set to any of the following:

1 NW_CONN_TYPE

4 NW_PBURST

8 NW_VERSION

15 NW_TRAN_TYPE

0x8000 NW_SESSION_ID

## NCP Calls

None

## See Also

**NWCCGetConnInfo, NWGetConnectionInformation**

# NWGetConnListFromObject

Returns a list of connection numbers a specified object has on a given server
**Local Servers:** blocking
**Remote Servers:** blocking
**NetWare Server:** 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetConnListFromObject (
   NWCONN_HANDLE   conn,
   nuint32         objID,
   nuint32         searchConnNum,
   pnuint16        connListLen,
   pnuint32        connList);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetConnListFromObject
  (connHandle : NWCONN_HANDLE;
   objID : nuint32;
   searchConnNum : nuint32;
   pConnListLen : pnuint16;
   pConnList : pnuint32
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the server to find connection numbers for.

*objID*

(IN) Specifies the object ID for which to get a list of connection numbers.

*searchConnNum*

(IN) Specifies the connection number to start searching from.

*connListLen*

(OUT) Points to a return buffer containing the number of connections

in the *connList* parameter.

*connList*

(OUT) Points to a return buffer containing up to 125 connection numbers.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x89FB | INVALID_PARAMETERS |

## NCP Calls

0x2222 23 31 Get Connection List From Object

# NWGetDefaultConnectionID (obsolete 9/97)

Returns the default connection handle of the current session but is now obsolete. Call the **NWGetDefaultConnRef** and **NWGetNearestDSConnRef** functions instead.

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetDefaultConnectionID
  (NWCONN_HANDLE N_FAR *  conn);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetDefaultConnectionID
  (Var pConnHandle : NWCONN_HANDLE
) : NWCCODE;
```

## Parameters

*conn*

(OUT) Points to the default connection handle.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |

## Remarks

### Remarks

The NetWare server containing the current directory for the connection is the default NetWare server.

The default connection handle corresponds to one of the following.

If the current drive is a network drive, the default connection handle is the server to which the drive maps.

If the current drive is not a network drive, the default connection handle is the server to which the workstation first logged in (also called primary server).

If the first and second conditions fail, the default server is the first server in the connection list for the shell (happens if the workstation is logged out of the primary server).

### NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 22 Get Station's Logged Info (old)
0x2222 23 28 Get Station's Logged Info
0x2222 104 1 Ping for NDS NCP

### See Also

**NWCCGetPrimConnRef**, **NWGetPreferredConnName**

# NWGetDefaultConnRef

Returns the default connection reference of the current session
**Local Servers:** blocking
**Remote Servers:** blocking
**NetWare Server:** 2.2, 3.11, 3.12, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95
**Service:** Connection

## Syntax

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

N_EXTERN_LIBRARY (NWCCODE) NWGetDefaultConnRef (
   pnuint32   pConnReference);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetDefaultConnectionID (
   Var pConnReference : nuint32
) : NWCCODE;
```

## Parameters

*pConnReference*
(OUT) Points to the default connection reference.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x880F | NO_CONNECTION_TO_SERVER |

## Remarks

The NetWare server containing the current directory for the connection is

The NetWare server containing the current directory for the connection is the default NetWare server.

The default connection reference corresponds to one of the following.

If the current drive is a network drive, the default connection reference is the server to which the drive maps.

If the current drive is not a network drive, the default connection reference is the server to which the workstation first logged in (also called primary server).

If the first and second conditions fail, the default server is the first server in the connection list for the shell (which occurs if the workstation is logged out of the primary server).

## NCP Calls

None

## See Also

**NWCCGetPrimConnRef**

# NWGetInetAddr

Returns the internet address of the *connNum* parameter on the specified NetWare server for the specified connection

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetInetAddr (
   NWCONN_HANDLE         connHandle,
   NWCONN_NUM            connNum,
   NWINET_ADDR N_FAR  *pInetAddr);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetInetAddr
  (connHandle : NWCONN_HANDLE;
   connNum : NWCONN_NUM;
   Var pInetAddr : NWINET_ADDR
) : NWCCODE;
```

## Parameters

*connHandle*

(IN) Specifies the NetWare server connection handle associated with the *connNum* parameter.

*connNum*

(IN) Specifies the connection number of the station whose internetwork address is to be returned.

*pInetAddr*

(OUT) Points to the internetwork address of the *connNum* parameter (10 bytes).

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x89FF | Failure |

## Remarks

An internetwork address consists of the *networkAddr* and *netNodeAddr* fields. (The *netNodeAddr* field is the physical address of the LAN board for the workstation.) The internetwork address uniquely identifies a workstation throughout an internetwork. The address can be used to send packets directly to the workstation.

To print the contents of the *pInetAddr* parameter, swap each byte by calling the **NWLongSwap** function on the *networkAddr* field, the **NWWordSwap** function on the first 2 bytes of the *netNodeAddr* field , and the **NWLongSwap** function on bytes 2 to 5 of the *netNodeAddr* field. Otherwise, the *pInetAddr* parameter appears in the format other functions expect.

See Listing Internetwork Addresses: Example.

## NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 19 Get Internet Address
0x2222 23 26 Get Internet Address (new)

# NWGetMaximumConnections (obsolete 9/97)

Returns the maximum number of connections available for the requesting workstation but is now obsolete. Call the **NWCCGetNumConns** function instead.

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

void N_API NWGetMaximumConnections (
   pnuint16   maxConns);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Procedure NWGetMaximumConnections
  (pMaxConns : pnuint16
);
```

## Parameters

*maxConns*

(OUT) Points to the maximum number of connections available.

## Return Values

None

## Remarks

Under the OS/2 operating system, the maximum number of connections (by default) is eight. However, the number can be increased to a maximum of 32 per workstation (see *NetWare Requester for OS/2*, Appendix B, part number 100-000492-001).

Under NETX, the maximum number of connections is eight.

Under VLMs, the maximum number of connections can be set in the NET.CFG file and cannot exceed 50.

### *NCP Calls*

None

# NWGetNearestDirectoryService (obsolete 9/97)

Returns a connection to the nearest Directory Services NetWare server (distance is determined by clock ticks) but is now obsolete. Call the **NWGetNearestDSConnRef** and **NWCCOpenConnByRef** functions instead.

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

```
#include <nwnet.h>
see also
#include <nwndscon.h>

NWCCODE N_API NWGetNearestDirectoryService
  (NWCONN_HANDLE N_FAR *   conn);
```

## Pascal Syntax

```
Function NWGetNearestDirectoryService
  (Var conn : NWCONN_HANDLE
) : NWCCODE;
```

## Parameters

*conn*

(OUT) Points to a connection handle to the nearest NDS server in the connection table.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |

## Remarks

### Remarks

If no NDS servers are found, INVALID_CONNECTION is returned.

**NWGetNearestDirectoryService (obsolete 9/97)** does not guarantee a licensed connection to a NetWare server.

### NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 22 Get Station's Logged Info (old)
0x2222 23 28 Get Station's Logged Info
0x2222 104 01 Ping for NDS NCP

### See Also

**NWCCGetConnInfo**, **NWCCOpenConnByName**, **NWCCOpenConnByRef**, **NWCCScanConnRefs**, **NWGetPreferredConnName**, **NWSetPreferredDSTree**

# NWGetNearestDSConnRef

Returns a connection reference to the nearest existing connection for a Directory Services NetWare server (distance is determined by clock ticks)

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

N_EXTERN_LIBRARY (NWCCODE) NWGetNearestDSConnRef (
   pnuint32   connRef);
```

## Pascal Syntax

```
Function NWGetNearestDirectoryService (
   connRef : pnuint32
) : NWCCODE;
```

## Parameters

*connRef*

(OUT) Points to a connection reference for the nearest NDS server in the connection table.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8846 | NWE_DS_NO_CONN |

## Remarks

If no NDS servers are found, NWE_DS_NO_CONN is returned.

If no NDS servers are found, NWE_DS_NO_CONN is returned.

### NCP Calls

None

### See Also

**NWCCGetAllConnRefInfo**, **NWCCGetConnRefInfo**, **NWCCOpenConnByRef**, **NWCCScanConnRefs**, **NWGetPreferredConnName**, **NWGetPreferredDSServer (obsolete 6/96)** , **NWSetPreferredDSTree**

# NWGetObjectConnectionNumbers

Returns a list of server connection numbers for clients logged in with the specified object name and type

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetObjectConnectionNumbers (
   NWCONN_HANDLE       conn,
   pnstr8              objName,
   nuint16             objType,
   pnuint16            numConns,
   NWCONN_NUM N_FAR   *connList,
   nuint16            maxConns);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetObjectConnectionNumbers
  (connHandle : NWCONN_HANDLE;
   pObjName : pnstr8;
   objType : nuint16;
   pNumConns : pnuint16;
   Var pConnHandleList : NWCONN_NUM;
   maxConns : nuint16
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the NetWare server connection handle.

*objName*

(IN) Points to the bindery object name of the object whose network server connection numbers are being obtained.

*objType*

(IN) Specifies the bindery object type of the object whose network

server connection numbers are being returned.

*numConns*

(OUT) Points to the number of server connections for the specified object.

*connList*

(OUT) Points to an array of the server connection numbers for the specified object.

*maxConns*

(IN) Specifies the size of the connection list array (maximum length=50).

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x8996 | SERVER_OUT_OF_MEMORY |
| 0x89F0 | WILD_CARD_NOT_ALLOWED |
| 0x89FC | NO_SUCH_OBJECT (2.x servers only) |
| 0x89FE | DIRECTORY_LOCKED |
| 0x89FF | HARDWARE_FAILURE |

## Remarks

If no client is logged in using the specified object name and object type, the list length returned by the server is set to zero.

The *numConns* parameter value is used to index the array pointed to by the *connList* parameter.

If an invalid object name or object type is passed on a 3.x or 4.x server, **NWGetObjectConnectionNumbers** will return SUCCESS and the *numConns* parameter will be zero indicating there are no connections with the server.

If an invalid object name or object type is passed on a 2.x server, **NWGetObjectConnectionNumbers** will return NO_SUCH_OBJECT.

### NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 21 Get Object Connection List
0x2222 23 27 Get Object Connection List (if 3.11 server)

# NWGetPrimaryConnectionID (obsolete 6/96)

Returns the primary network server connection handle for a workstation but is now obsolete. Call **NWCCGetPrimConnRef** instead.

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetPrimaryConnectionID
  (NWCONN_HANDLE N_FAR  *conn);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWGetPrimaryConnectionID
  (Var pConnHandle : NWCONN_HANDLE
) : NWCCODE;
```

## Parameters

*conn*

(OUT) Points to the primary server connection handle for the workstation.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x8831 | PRIMARY_CONNECTION_NOT_SERVER |

### *Remarks*

The initial primary connection handle is the ID of the server from which the login script is read. It is set by utilities such as Login.

### *NCP Calls*

None

# NWGetTaskInformationByConn

Returns information about the active tasks a specified connection has
**Local Servers:** blocking
**Remote Servers:** blocking
**NetWare Server:** 2.2, 3.11, 3.12, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
**Service:** Connection

## Syntax

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

NWCCODE N_API NWGetTaskInformationByConn (
   NWCONN_HANDLE           conn,
   NWCONN_NUM              connNum,
   CONN_TASK_INFO N_FAR  *taskInfo);
```

## Pascal Syntax

```
#include <nwmisc.inc>

Function NWGetTaskInformationByConn
  (conn : NWCONN_HANDLE;
   connNum : NWCONN_NUM;
   Var taskInfo : CONN_TASK_INFO
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the NetWare server connection handle. NWCONN_HANDLE is equivalent to nuint16.

*connNum*

(IN) Specifies the connection number of the logged-in object for which to get task information. NWCONN_NUM is equivalent to nuint16.

*taskInfo*

(OUT) Points to theCONN_TASK_INFO structure.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x8996 | SERVER_OUT_OF_MEMORY |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FD | BAD_STATION_NUMBER |
| 0x88FE | Unknown Packet Format |

### NCP Calls

0x2222 23 218 Get Connection's Task Information (2.x)
0x2222 23 234 Get Connection's Task Information (3.x-4.x)

# NWIsIDInUse (obsolete 6/96)

Returns TRUE if the specified connection handle is in use but is now obsolete

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWIsIDInUse
   (NWCONN_HANDLE    conn);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWIsIDInUse
   (connHandle : NWCONN_HANDLE
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the NetWare server connection handle to check for use.

## Return Values

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

| | |
|---|---|
| 0x0000 | ID is in use |
| non zero | ID is not in use |

## Remarks

**NWIsIDInUse (obsolete 6/96)** is meaningless when VLMs are running.

### *NCP Calls*

None

# NWRequest

Passes an NCP request to the server
**NetWare Server:** 2.2, 3.11, 3.12, 4.x
**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
**Service:** Connection

### Syntax

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

NWCCODE N_API NWRequest (
   NWCONN_HANDLE         conn,
   nuint16               function,
   nuint16               numReqFrags,
   NW_FRAGMENT N_FAR    *reqFrags,
   nuint16               numReplyFrags,
   NW_FRAGMENT N_FAR   *replyFrags);
```

### Pascal Syntax

```
#include <nwmisc.inc>

Function NWRequest
  (conn : NWCONN_HANDLE;
   functionID : nuint16;
   numReqFrags : nuint16;
   Var reqFrags : NW_FRAGMENT;
   numReplyFrags : nuint16;
   Var replyFrags : NW_FRAGMENT
) : NWCCODE;
```

### Parameters

*conn*

(IN) Specifies the NetWare server connection handle.

*function*

(OUT) Receives the NCP function number.

*numReqFrags*

(OUT) Receives the number of fragments pointed to by the *reqFrags* field (maximum is five).

*reqFrags*

(OUT) Points to the list of request fragments.

*numReplyFrags*

(OUT) Receives the number of fragments pointed to by the *replyFrags* field (maximum is five).

*replyFrags*

(OUT) Points to the NW_FRAGMENT structure containing the list of reply fragments.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| other | Error from NCP or Requester |

## Remarks

The request and reply fragment buffers should not overlap. The total length of the request and reply fragments should not exceed 576 bytes.

## NCP Calls

None

# NWSetPrimaryConnectionID (obsolete 6/96)

Sets or resets the primary connection handle but is now obsolete. Call **NWCCSetPrimConn** instead.

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows 95

**Service:** Connection

## Syntax

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

NWCCODE N_API NWSetPrimaryConnectionID
  (NWCONN_HANDLE   conn);
```

## Pascal Syntax

```
#include <nwconnec.inc>

Function NWSetPrimaryConnectionID
  (connHandle : NWCONN_HANDLE
) : NWCCODE;
```

## Parameters

*conn*

(IN) Specifies the NetWare server connection handle to set as the primary server.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |

## Remarks

The initial primary connection handle is the ID of the server from which

the login script is read. The primary connection handle is set by utilities such as Login.

**CAUTION: To avoid unpredictable results, call NWSetPrimaryConnectionID (obsolete 6/96) cautiously in applications that depend on the primary connection handle to point to the actual primary server.**

### NCP Calls

None

# NWUnlicenseConn (obsolete 6/96)

Unlicenses a connection but is now obsolete. Call **NWCCUnlicenseConn** instead.

**NetWare Server:** 2.2, 3.11, 3.12, 4.x

**Platform:** UNIX

**Service:** Connection

## Syntax

```
#include <nwconnec.h>

NWCCODE N_API NWUnlicenseConn(
   NWCONN_HANDLE   conn);
```

## Parameters

*conn*

(IN) Specifies the open connection handle needing the connection to be unlicensed.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x8815 | HANDLE_ALREADY_UNLICENSED |

## Remarks

When a connection is opened, it is automatically licensed. If the application opening the connection does not require a licensed connection, **NWUnlicenseConn (obsolete 6/96)** can be called to unlicense the connection. In the requester, **NWUnlicenseConn (obsolete 6/96)** will only unlicense the connection if there are no other open handles needing the licensed connection.

## NCP Calls

None

## *See Also*

**NWCCLicenseConn**

# SetConnectionCriticalErrorHandler

Specifies a function to handle connection timeout errors
**Local Servers:** blocking
**Remote Servers:** N/A
**SMP Aware:** No
**NetWare Server:** 4.x
**Platform:** NLM
**Service:** Connection

## Syntax

```
#include <nwconn.h>

int SetConnectionCriticalErrorHandler  (
   int   (func) (
      int   fileServerID,
      int   connection,
     int   err));
```

## Parameters

*func*

(IN) Specifies a custom function for handling connection timeout errors.

## Return Values

ESUCCESS or NetWare errors

## Remarks

The *func* parameter is a custom function for handling connection timeout errors. When a timeout occurs on a connection, **SetConnectionCriticalErrorHandler** is passed the *fileServerID*, *connection*, and *err* for the connection that failed. The error handling function should return the number of times to try to restore the connection.

## NCP Calls

None

# Connection:  Structures

# CONN_TASK

**Service:** Connection
**Defined In:** nwmisc.h and nwmisc.inc

### Structure

```
typedef struct {
   nuint16   taskNumber;
   nuint8    taskState;
} CONN_TASK;
```

### Pascal Structure

```
CONN_TASK = Record
    taskNumber : nuint16;
    taskState : nuint8
  End;
```

### Fields

*taskNumber*

Specifies the server task number for which information will be returned.

*taskState*

Specifies the state of the task at the time of the request.

### Remarks

The *taskState* field can have the following values:

0x00 Normal Task
0x01 TTS explicit transaction in progress
0x02 TTS implicit transaction in progress
0x04 Shared fine set lock in progress

# CONN_TASK_INFO

Returns information according to the*lockState* field
**Service:** Connection
**Defined In:** nwmisc.h and nwmisc.inc

### Structure

```
typedef struct {
   nuint16     serverVersion;
   nuint8      lockState;
   nuint16     waitingTaskNumber;
   nuint32     recordStart;
   nuint32     recordEnd;
   nuint8      volNumber;
   nuint32     dirEntry;
   nuint8      nameSpace;
   nuint16     dirID;
   nstr8       lockedName[256];
   nuint8      taskCount;
   CONN_TASK   tasks[256];
} CONN_TASK_INFO;
```

### Pascal Structure

```
CONN_TASK_INFO = Record
    serverVersion : nuint16;
    lockState : nuint8;
    waitingTaskNumber : nuint16;
    recordStart : nuint32;
    recordEnd : nuint32;
    volNumber : nuint8;
    dirEntry : nuint32;
    nameSpace : nuint8;
    dirID : nuint16;
    lockedName : Array[0..255] Of nstr8;
    taskCount : nuint8;
    tasks : Array[0..255] Of CONN_TASK
  End;
```

### Fields

*serverVersion*

Specifies the server version (NW_ constants in nwserver.h).

*lockState*

Specifies one of five lock states that are used internally to determine what information can be returned.

*waitingTaskNumber*

Specifies the task number returned when the *lockState* field has a nonzero value.

*recordStart*

Specifies the start address of physical record returned when the *lockState* field has a value of 1.

*recordEnd*

Specifies the end address of physical record returned when the *lockState* field has a value of 1.

*volNumber*

Specifies the volume number of physical record or file returned when the *lockState* field has a value of 1 or 2.

*dirEntry*

Specifies the directory entry of physical record or file returned when the *lockState* field has a value of 1 or 2 (valid only in 3.11 or higher).

*nameSpace*

Specifies the name space of locked file (valid only in 3.11 or higher).

*dirID*

Specifies the ID of the directory (valid only in 2.x).

*lockedName*

Specifies the name of the locked physical record, file, logical record, or semaphore.

*taskCount*

Specifies the number of tasks for which the CONN_TASK structure will be returned.

*tasks*

Specifies the CONN_TASK structure containing information for each task counted in the *taskCount* field.

## Remarks

The *lockState* field can have the following values:

0 Normal (connection free to run)
1 Connection waiting on a physical record lock
2 Connection waiting on a file lock
3 Connection waiting on a logical record lock
4 Connection waiting on a semaphore

The *nameSpace* field can have the following values:

0 NW_NS_DOS
1 NW_NS_MAC

```
2  NW_NS_NFS
3  NW_NS_FTAM
4  NW_NS_OS2
4  NW_NS_LONG
```

# CONN_USE

Returns connection usage statistics
**Service:** Connection
**Defined In:** nwconnec.h

### Structure

```
typedef struct (
   nuint32   systemElapsedTime;
   nuint8     bytesRead[6];
   nuint8     bytesWritten[6];
   nuint32   totalRequestPackets;
) CONN_USE;
```

### Pascal Structure

```
   CONN_USE = Record
    systemElapsedTime : nuint32;
    bytesRead : Array[0..5] Of nuint8;
    bytesWritten : Array[0..5] Of nuint8;
    totalRequestPackets : nuint32
  End;
```

### Fields

*systemElapsedTime*

Indicates how long the server has been up. Currently *systemElapsedTime* is returned in 18.2 ticks/second units. When it reaches 0xFFFFFFFF, *systemElapsedTime* wraps back to 0.

*bytesRead*

Indicates the number of bytes the associated connection has read.

*bytesWritten*

Indicates the number of bytes the associated connection has written.

*totalRequestPackets*

Indicates the number of requests the associated connection has made.

# CONNECT_INFO

Returns connection status information
**Service:** Connection
**Defined In:** nwconnec.h

### Structure

```
typedef struct
{
   NWCONN_HANDLE   connID;
   nuint16         connectFlags;
   nuint16         sessionID;
   NWCONN_NUM      connNumber;
   nuint8          serverAddr[12];
   nuint16         serverType;
   nstr8           serverName[C_SNAMESIZE];
   nuint16         clientType;
   nstr8           clientName[C_SNAMESIZE];
} CONNECT_INFO;
```

### Pascal Structure

```
Defined in nwconnec.inc

  CONNECT_INFO = Record
    connID : NWCONN_HANDLE;
    connectFlags : nuint16;
    sessionID : nuint16;
    connNumber : NWCONN_NUM;
    serverAddr : Array[0..11] Of nuint8;
    serverType : nuint16;
    serverName : Array[0..C_SNAMESIZE-1] Of nstr8;
    clientType : nuint16;
    clientName : Array[0..C_SNAMESIZE-1] Of nstr8
  End;
```

### Fields

*connID*

Indicates the connection handle associated with the information in this structure.

*connectFlags*

Indicates the flag whose values are defined:

| C | Pasc | Flag Name | Description |
|---|---|---|---|

| Values | Hexadecimal Values | | |
|---|---|---|---|
| 0x0001 | $0001 | CONNECTION_AVAILABLE | Indicates if the specified connection handle hasn't been allocated to a process at the workstation. |
| 0x0002 | $0002 | CONNECTION_PRIVATE | |
| 0x0004 | $0004 | CONNECTION_LOGGED_IN | Indicates if the client is logged in on the connection. |
| 0x0004 | $0004 | CONNECTION_LICENSED | |
| 0x0008 | $0008 | CONNECTION_BROADCAST_AVAILABLE | Indicates if broadcasts to other stations are available on the connection. |
| 0x0010 | $0010 | CONNECTION_ABORTED | Indicates if the connection was aborted. |
| 0x0020 | $0020 | CONNECTION_REFUSE_GEN_BROADCAST | Indicates if general broadcasts are not to be received on the connection. |
| 0x0040 | $0040 | CONNECTION_BROADCAST_DISABLED | Indicates if no broadcasts will be received on the connection. |
| 0x0080 | $0080 | CONNECTION_PRIMARY | Indicates if this |

| | | | |
|---|---|---|---|
| | | | connection handle is the workstation's primary connection with the network. |
| 0x0100 | $0100 | CONNECTION_NDS | Indicates if the connection is a Directory Services connection. |
| 0x0400 | $4000 | CONNECTION_PNW | |
| 0x8000 | $8000 | CONNECTION_AUTHENTICATED | Indicates if the connection is authenticated. |

*sessionID*

Indicates the current session ID. *sessionID* is only valid when VLMs are installed on the workstation. If NETX.EXE is being used, *sessionID* is always zero (0).

*connNumber*

Indicates the client's connection as seen from the NetWare server.

*serverAddr*

Indicates the Internet address consisting of the network number (first 4 bytes) and the physical node address (bytes 5-10).

*serverType*

Indicates the server's bindery object type.

*serverName*

Indicates the server's bindery name.

*clientType*

Indicates the client' bindery object type.

*clientName*

Indicates the client's bindery object name.

# NW_FRAGMENT

Fragments the request into the appropriate packet size
**Service:** Connection
**Defined In:** nwmisc.h and nwmisc.inc

### Structure

```
typedef struct {
   nptr      fragAddress;
   #if defined(N_PLAT_NLM)||defined(WIN32)
      nuint32   fragSize;
   #else
      nuint16   fragSize;
   #endif
} NW_FRAGMENT;
```

### Pascal Structure

```
NW_FRAGMENT = Record
  fragAddress : nptr;
  fragSize : nuint16
End;
```

### Fields

*fragAddress*

Points to where the fragment starts.

*fragSize*

Specifies the size of the fragment (under NLM and Windows platforms, of type nuint32).

# NWCCConnInfo

Returns the specified information for a given connection
**Service:** Connection
**Defined In:** nwclxcon.h and nwclxcon.inc

### Structure

```
typedef struct
   nuint              authenticationState;
   nuint              broadcastState;
   nuint32            connRef;
   nstr               treeName[NW_MAX_TREE_NAME_LEN];
   nuint              connNum;
   nuint32            userID;
   nstr               serverName[NW_MAX_SERVER_NAME_LEN];
   nuint              NDSState;
   nuint              maxPacketSize;
   nuint              licenseState;
   nuint              distance;
   NWCCVersion        serverVersion;
   pNWCCTranAddr      tranAddr;
} NWCCConnInfo;
```

### Pascal Structure

```
NWCCConnInfo = Record
    authenticationState :  nuint;
    broadcastState :  nuint;
    connRef :  nuint32;
    treeName : Array[0..NW_MAX_TREE_NAME_LEN-1] Of nstr;
    connNum :  nuint;
    userID :  nuint32;
    serverName : Array[0..NW_MAX_SERVER_NAME_LEN-1] Of nstr;
    NDSState : nuint;
    maxPacketSize : nuint;
    licenseState : nuint;
    distance : nuint;
    serverVersion : NWCCVersion;
   {$IFDEF NWCC_INFO_VERSION_2}
      tranAddr : pNWCCTranAddr;
   {$ENDIF}
  End;
```

### Fields

*authenticationState*

Indicates the Novell Directory Services™ authenticated state of the

specified connection:

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0000 | $0000 | NWCC_AUTHENT_STATE_NONE |
| 0x0001 | $0001 | NWCC_AUTHENT_STATE_BIND |
| 0x0002 | $0002 | NWCC_AUTHENT_STATE_NDS |

*broadcastState*

Indicates the message broadcast state of the specified connection:

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0000 | $0000 | NWCC_BCAST_PERMIT_ALL |
| 0x0001 | $0001 | NWCC_BCAST_PERMIT_SYSTEM |
| 0x0002 | $0002 | NWCC_BCAST_PERMIT_NONE |
| 0x0003 | $0003 | NWCC_BCAST_PERMIT_POLL |

*connRef*

Indicates the connection reference for the specified connection handle.

*treeName*

Indicates the tree name of the specified connection if attached to Directory Services. It has a maximum length of NWA_MAX_TREE_NAME_LEN.

*connNum*

Indicates the connection number for the specified connection.

*userID*

Indicates the user for the specified connection.

*serverName*

Indicates the name of the server the connection is attached to. It has a maximum length of NWA_MAX_SERVER_NAME_LEN.

*NDSState*

Indicates if the connection supports Novell Directory Services.

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0000 | $0000 | NWCC_NDS_NOT_CAPABLE |
| 0x0001 | $0001 | NWCC_NDS_CAPABLE |

*maxPacketSize*

Indicates the maximum length of an Internet packet that can be supported by this connection.

*licenseState*

Indicates if the connection is licensed:

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0000 | $0000 | NWCC_NOT_LICENSED |
| 0x0001 | $0001 | NWCC_CONNECTION_LICENSED |
| 0x0002 | $0002 | NWCC_HANDLE_LICENSED |

*distance*

Indicates distance in milliseconds to the given server (55 milliseconds = 1 tick).

*serverVersion*

Points to the NWCCVersion structure returning the NetWare version.

*tranAddr*

Points to the NWCCTranAddr structure returning the type.

## Remarks

The *treeName* structure field is used to give the tree name of a particular connection in functions such as **NWCCGetAllConnRefInfo** and **NWCCGetConnRefInfo**. You must strip off the trailing `_' characters (that are padding the tree name out to the maximum length) to get a matching valid tree name. Other functions that depend on a valid tree name already strip the `_' characters.

# NWCCTranAddr

Defines the transport address for the specified connection
**Service:** Connection
**Defined In:** nwclxcon.h

### Structure

```
typedef struct
   nuint32        type;
   nuint32        len;
   pnuint8        buffer;
} NWCCTranAddr;
```

### Pascal Structure

```
Defined in nwclxcon.inc

NWCCTranAddr = Record
    tranType : nuint32;
    len : nuint32;
    buffer : pnuint8
  End;
```

### Fields

*type*

(IN/OUT) Specifies the type of the transport address:

| C Value | Pascal Value | Value Name |
|---|---|---|
| 0x0001 | $0001 | NWCC_TRAN_TYPE_IPX |
| 0x0002 | $0002 | NWCC_TRAN_TYPE_UDP |
| 0x0003 | $0003 | NWCC_TRAN_TYPE_DDP |
| 0x0004 | $0004 | NWCC_TRAN_TYPE_ASP |

*len*

(IN/OUT) Specifies the length of the buffer to hold the transport address upon input. Specifies the amount of the buffer that was actually used upon output.

*buffer*

(OUT) Points to a buffer containing the transport address.

### Remarks

### *Remarks*

If the value returned in the *len* field is greater than the original value passed to the *len* field, the returned value specifies the total length of the buffer that is needed to return all the information.

# NWCCVersion

Defines the NetWare server version of the connection
**Service:** Connection
**Defined In:** nwclxcon.h

### Structure

```
typedef struct
   nuint   major;
   nuint   minor;
   nuint   revision;
} NWCCVersion;
```

### Pascal Structure

```
Defined in nwclxcon.inc

 NWCCVersion = Record
    major : nuint;
    minor : nuint;
    revision : nuint
  End;
```

### Fields

*major*

Indicates the major version of NetWare. For example, *major* will be 4 for NetWare 4.1.

*minor*

Indicates the minor version of NetWare. For example, *minor* will be 12 for NetWare 3.12.

*revision*

Indicates an interim release number.

# NWINET_ADDR

Returns the internet address for the specified connection
**Service:** Connection
**Defined In:** nwclxcon.h

### Structure

```
typedef struct
{
   nuint8    networkAddr[4];
   nuint8    netNodeAddr[6];
   nuint16   socket;
   nuint16   connType;
} NWINET_ADDR;
```

### Pascal Structure

```
 NWINET_ADDR = Record
    networkAddr : Array[0..3] Of nuint8;
    netNodeAddr : Array[0..5] Of nuint8;
    socket : nuint16;
    connType : nuint16 (*0=not in use,2=NCP,3=AFP *)
  End;
```

### Fields

*networkAddr*

Indicates the network address.

*netNodeAddr*

Indicates the network node address.

*socket*

Indicates the network socket.

*connType*

Indicates the connection type. Used for 3.11 and above only.

0=not in use
2=NCP
3=AFP

# Connection Number and Task Management

# Connection Number and Task Management:  Guides

The Connection Number and Task Management Services are available only for managing connections in NLM applications. Use the Connection Services to get and free connections using non-NLM applications.

## Connection Number and Task Management: Task Guide

Specifying a Connection Number

Logging In

Intervening on an Established Connection

Using Connections

Doing Work on a Single Connection

Multiple Thread Groups on a Single Connection

Doing Work by Proxy

Using Connection 0

Using the Number of an Already Logged-In Workstation

Allocating a New Connection Number and Logging In

Allocating One or More Tasks

## Connection Number and Task Management: Concept Guide

Overview of Connections and Tasks

Single Connection, Many Users

Current Connection and Task

NLM Applications and Connections

Tasks in NetWare

Remote and Local Connections

Connection:  Functions

# Connection Number and Task Management:  Tasks

**NOTE:** Connection Number and Task Management Services provide connection functions for NLM development only.

## Allocating a New Connection Number and Logging In

An NLM can allocate a new connection number and then log in one of its users on that connection. NLM applications that use connections in this way fit the same profile as those that **take over** a user's connection. It is another way that an NLM can do work for a user and charge the user for resource consumption and preserve the user's trustee rights.

An NLM can allocate a new connection number with one of the following functions:

**AttachByAddress** or **AttachToFileServer**

**SetCurrentConnection**

When an NLM calls one of these functions, it is attached to the server but not authenticated (not logged-in). Therefore, it has very limited access rights. Your NLM can then log in a user on that connection by calling **LoginObject**, passing it the user's Directory or Bindery name, object type, and password. (If you don't know the password, you can pass in LOGIN_WITHOUT_PASSWORD.) The access rights of your NLM on that connection are those of the particular user.

When using **SetCurrentConnection**, you may set any connection number. However, on remote servers, you may set only those that your NLM has logged in on with **LoginToFileServer** (Connection Services).

Allocating a Connection Number and Logging In: Example

**Parent Topic:**

Connection Number and Task Management:  Guides

## Allocating One or More Tasks

If you want to specify a task number other than the one your current thread

group has, you can do so with **SetCurrentTask**. It sets the current task to the value you pass it as *taskNumber*.

You can also use **SetCurrentTask** to allocate one new task number for the current thread group. If you pass -1 as the *taskNumber*, it allocates a new task number; otherwise, it sets the current task to the value you pass it.

If you need more than one task number, use **AllocateBlockOfTasks**. This function returns the first task number. If it fails, it returns zero

When you are finished using the tasks that were allocated by either **AllocateBlockOfTasks** or **SetCurrentTask**, always call **ReturnBlockOfTasks** to free the task numbers before unloading.

> **NOTE:** The **setupTasks** function in Allocating One or More Tasks: Example allows for single-task and multiple-task scenarios. It calls **AllocateBlockOfTasks** if *NumberOfTasks* is greater than 1, and **SetCurrentTask** with *taskNumber* as -1 if *NumberOfTasks* is less than one. When **SetCurrentTask** is passed -1, it allocates a new task. If another value had been passed, it would have set the current task to that number. **AllocateBlockOfTasks**, on the other hand, simply allocates a set of tasks without setting the current task.

Allocating One or More Tasks: Example

**Parent Topic:**

Connection Number and Task Management: Guides

# Doing Work by Proxy

When an NLM does work **by proxy**, it makes requests to the server under the connection number of the client on whose behalf it is making the request.

An NLM can do work by proxy in a couple of ways. It can either get a connection and log an object of its choosing in on that connection, or it can temporarily **take over** the client's existing connection to the local server.

Logging In

**Parent Topic:**

Connection Number and Task Management: Guides

# Doing Work on a Single Connection

One advantage of using a single connection is that your does not take up a large number of connections on a server. A disadvantage is that if all clients are being served on a single connection, there is no way to determine which client requested the work. Consequently, there is no way to testing whether

the client has the trustee rights necessary to perform that request. Nor is there a way to charge the client for using server resources.

An NLM can use a single connection to do all its work on behalf of all its clients. It could use connection 0 or it could use a connection that it has obtained by calling **NWLoginToFileServer** or **NWAttachToFileServer**.

**Parent Topic:**

Connection Number and Task Management: Guides

# Intervening on an Established Connection

Call **DisableConnection** to prevent the server from filling any requests other than those your NLM originates until your NLM is finished using a connection. Be careful when calling this function since the client using the specified connection is temporarily disabled.

**Parent Topic:**

Connection Number and Task Management: Guides

# Logging In

In addition to being able to change your current connection and task number, you can log an object in on any connection that your NLM has allocated.

Your NLM can get a connection to a server, local or remote, and then log in the client on that connection with **LoginObject**.

**DisableConnection** allows your NLM to disable a connection for any object's requests except those originated by your NLM until it is finished using the connection.

**Parent Topic:**

Connection Number and Task Management: Guides

# Specifying a Connection Number

When making a request, an NLM can specify three types of connection numbers:

connection 0

the number of an already logged-in workstation

a new connection number (obtained from a server)

**Parent Topic:**

Connection Number and Task Management: Guides

# Using Connection 0

Because connection 0 is server-equivalent (an extension of the server), if an NLM does work for its clients as connection 0, the server is not able to tell which client it is servicing. For this reason, and because connection 0 has unrestricted access to services, only NLM applications or their threads that fit the following profile should do work as connection 0:

They are not depending on NetWare Accounting to track their users' consumption of resources.

Their access to NetWare (trustee rights) never needs to be restricted.

Monitor and control programs fit this profile.

**Parent Topic:**

Connection Number and Task Management: Guides

# Using Connections

Because NLM applications are loaded into server memory alongside the server, they are granted special access to the server through **connection zero (0)** on **file server ID zero (0)**. Connection 0 denotes that the connection is **local** to the server, as opposed to a connection sending requests from a remote server or workstation.

See the figure Remote and Local NLM Applications for an illustration of local and remote connections. The server where the NLM is loaded is **file server ID 0**. Connection 0 is only valid when the thread group's current file server ID is 0.

> **NOTE:** NLM applications using connection 0 create an unsecure environment. The server assumes that, since the request is on connection 0, it need not worry about security. As you can see, NLM applications are trusted partners working with the NetWare OS.

As trusted partners, NLM applications share in the control of certain operating system resources, such as the CPU, file system, and connections. This interaction in NLM applications the ability to manipulate connections and tasks in ways not available to applications running on workstations.

**Parent Topic:**

Connection Number and Task Management: Guides

# Using the Number of an Already Logged-In Workstation

Your NLM may specify the connection number of a client so that its access is limited to the trustee rights of the object logged in on that connection.

Most client-server programs that use the client's connection number conform to a profile opposite that for connection 0:

They need charge the client for services consumed by the NLM on the client's behalf.

They need to restriction the client's access to network resources. For example, only the supervisor has the right to close the Bindery. But if an NLM were doing work for a user that is not supervisor as connection 0, it would execute a request from that user to close the Bindery. Also, you wouldn't want all the users of your database NLM to have access to the entire database.

An NLM that is using a client's connection number to access the file system should do one of the following two things to ensure it has a unique connection/task number pair:

1. To avoid accessing the file system at the same time as your client workstation or other NLM applications, use **DisableConnection** to temporarily reserve the workstation's connection number solely for the use of the NLM. Call **EnableConnection** to reverse the effect of **DisableConnection**. Be careful when calling this function since the client using the specified connection is temporarily disabled.

2. Allocate a new task and set that to be your current task number using **SetCurrentTask**.

**Parent Topic:**

Connection Number and Task Management: Guides

# Connection Number and Task Management: Examples

**NOTE:** Connection Number and Task Management Services provide connection functions for NLM development only.

## Allocating a Connection Number and Logging In: Example

This example function shows the process of allocating a new connection number and logging in a user on it. Although it shows **AttachToFileServer** allocating a connection on the local server, you would use this function or **AttachByAddress** the same way to get a connection to a remote server (however, **AttachByAddress** uses Directory addressing information instead of the server's name). Both these functions are in the Connection Services group.

After it allocates a connection, the **setupConnection** example function obtains the information it needs to log in a user with **LoginObject**. It uses **GetCurrentConnection** to retrieve the current connection:

```
if(rc = LoginObject(GetCurrentConnection(), objectName,
   OT_USER, password))
```

The **objectName** and **password** must be in upper case letters. Notice that **NWLstrupr** is called to ensure the proper upper casing for the locale.

This function could have also used **SetCurrentConnection** to allocate an unused connection number. It would have passed -1 as the **connectionNumber**. This would have allocated a connection number and made it the current thread group's current connection. Had it passed another value besides -1, **SetCurrentConnection** would have made that the thread group's current connection.

See Allocating a New Connection Number and Logging Infor further information about this example code.

```
void setupConnection()
{
   int  i;
   int  rc;
   char *ptr;
   char serverName[48];
   char objectName[48];
   char password[48];
```

```
    WORD fileServerID;
    GetFileServerName(0, serverName);
    if(rc = AttachToFileServer( serverName, &fileServerID ))
    {
       printf("Error %d attaching to server %s\n", rc, serverName);
    }
    printf("Enter object name: ");
    scanf("%s", objectName);
    if(!(ptr = NWLstrupr(objectName)))
       printf("Error in NWLstrupr\n");
    i = 0;
    memset(password, 0, 48);
    printf("Enter password (or Return for NULL): ");
    while((rc = getch()) > (int) ' ')
       password[i++] = rc;
    printf("\n");
    if(!(ptr = NWLstrupr(password)))
       printf("Error in NWLstrupr\n");
    if(rc = LoginObject(GetCurrentConnection(), objectName, OT_USER,
       password))
    {
       printf("Error %d logging in to server %s\n", rc, serverName);
       cleanup(0);
    }
    printf("Logged into server %s successfully\n\n", serverName);
}
```

**Parent Topic:**

Connection Number and Task Management: Guides

# Allocating One or More Tasks: Example

The following function example allocates single or multiple tasks. See Allocating One or More Tasks for additional information about this code.

```
#define    MAXIMUM_NUMBER_OF_TASKS         16
int NumberOfTasks;
int BeginningTaskNumber;
int TaskArray[MAXIMUM_NUMBER_OF_TASKS];
      .
      .
      .
void setupTasks()
{
   if(NumberOfTasks > 1)
   {
      if(!(BeginningTaskNumber =
           AllocateBlockOfTasks(NumberOfTasks)))
      {
```

```
      printf("Error %d allocating block of tasks\n",
      BeginningTaskNumber);
   }
}
else
{
   if((BeginningTaskNumber = SetCurrentTask(-1)) == -1)
   {
      printf("Error %d allocating task\n",
              BeginningTaskNumber);
   }
   .
   .
   .
if(rc = ReturnBlockOfTasks(BeginningTaskNumber,
        NumberOfTasks))
   printf("Error %d returning tasks\n", rc);
else
   printf("Cleaned up tasks successfully.\n");
```

**Parent Topic:**

Connection Number and Task Management: Guides

# Connection Number and Task Management:  Concepts

**NOTE:** Connection Number and Task Management provide connection functions for NLM development only.

## Connection Number and Task Management Functions

The descriptions of these functions in Connection Number and Task Management:  Functions use the terms station, connection, and connection number interchangeably.

*Table auto. Connection Number and Task Management Functions*

| Function | Purpose |
|---|---|
| **AllocateBlockOfTasks** | Returns a set of unique task numbers for the exclusive use of the requesting NLM. |
| **CheckIfConnectionActive** | Checks if the specified connection number is being used by a file access. |
| **DisableConnection** | Temporarily prevents the server from servicing any requests (except requests made by the calling NLM) for the specified connection number. Be careful when calling this function since the client using the specified connection is temporarily disabled.. |
| **EnableConnection** | Reverses the effect of **DisableConnection**. |
| **GetCurrentConnection** | Returns the calling thread group's current connection number. |
| **GetCurrentFileServerID** | Returns the calling thread group's current file server ID. |
| **GetCurrentTask** | Returns the calling thread group's current task number. |
| **LoginObject** | Logs in the specified object to the specified connection number on the current file server ID. |
| **LogoutObject** | Logs out the object logged-in on the specified connection number on the thread group's current file server ID. |

| | |
|---|---|
| **ReturnBlockOfTasks** | Frees the block of task numbers allocated with **AllocateBlockOfTasks** or **SetCurrentTask**. |
| **ReturnConnection** | Frees a connection the NLM had allocated. |
| **SetCurrentConnection** | For the current thread group, changes the current connection number. Can also be used to allocate a new connection number. |
| **SetCurrentFileServerID** | Sets the current file server's ID. |
| **SetCurrentTask** | Sets the calling thread group's current task number, or allocates a new task by passing in -1. |

**Parent Topic:**

Connection Number and Task Management: Guides

# Current Connection and Task

In NLM development, current connections and current tasks are part of a thread group's context (information that lets the CPU pick up where it left off when that thread was swapped out). An NLM has the ability to change the connection or task number of the current thread (the one being processed by the CPU).

For example, if you had an NLM that serves many clients, you might want to use a single thread to do work for a number of them in succession. After doing some work for one client, your NLM could call **SetCurrentConnection** to change the connection number of that thread to a different client's number. (This would set the current connection for the entire thread group.) Then, after doing some work for that client, your NLM could set the current connection to that of another client, and so on through the list of its clients.

You could change task numbers in the same manner with **SetCurrentTask** if you wanted to serve many or all of your clients on a single connection. You would spawn as many thread groups as you had clients and then set a different task number for each one. For example, if you had 10 clients and the connection you are using to service them is 52, you would allocate 10 task numbers for Connection 52. Say the server allocated you tasks 1 to 10. The connection/task number of the first thread group would be 52/1, the second thread group would be 52/2, and so on.

**Parent Topic:**

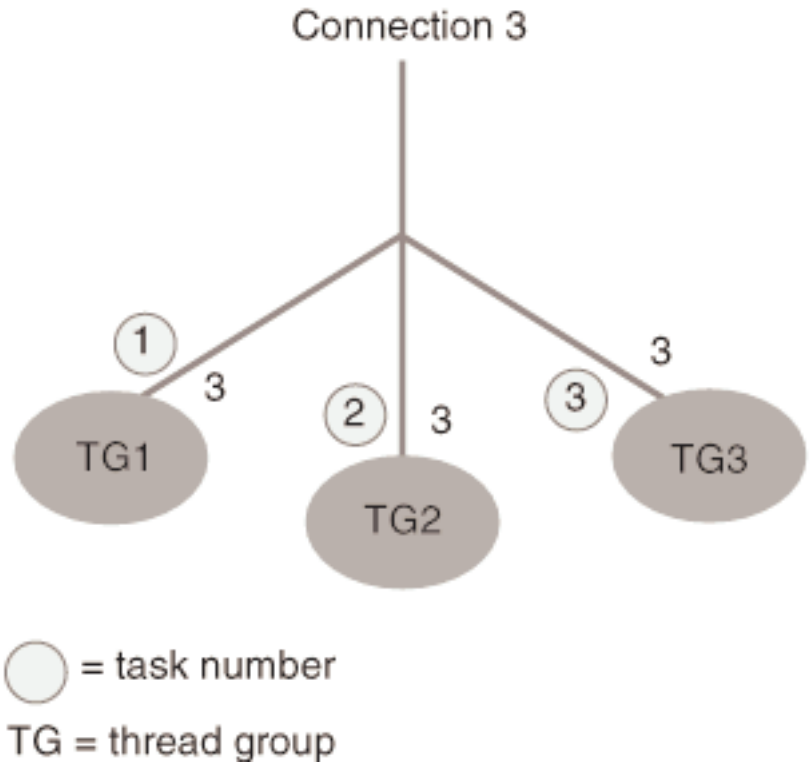
Connection Number and Task Management: Guides

# Multiple Thread Groups on a Single Connection

# Multiple Thread Groups on a Single Connection

You could give multiple thread groups the same **connectionNumber** with **SetCurrentConnection**. Then, with **SetCurrentTask**, you could assign each of them a different **taskNumber** and have each of them perform a different task for a single user. Multiple threads working a single connection can accomplish the work more efficiently than if a single thread were doing every single task involved. This is because, while a thread is waiting for information (blocking), other threads are running.

In the following figure, three thread groups---TG 1, TG 2, and TG 3---each have their connection set to 3 and each has a different task number. TG 1 is task 1, TG 2 is task 2, and so on. In this way, each thread group has a unique connection/task pair that is part of its own context.




To reduce the danger of duplicating a connection/task pair, call **AllocateBlockOfTasks** to get a unique set of tasks for that connection. This way you can use the server to keep track of task numbers.

A single connection works well for control and monitor types of programs, where rights and accounting are not issues. But, if you want to preserve your client's rights or track the client's use of resources, you will want to do work for them by proxy.

**Parent Topic:**

Connection Number and Task Management: Guides

# NLM Applications and Connections

An NLM obtains connections to a remote server in the same manner that a workstation does. An NLM also can get additional connections to its local server (besides connection 0).

To request a connection, an NLM uses the same functions that workstations do, **NWLoginToFileServer** or **NWAttachToFileServer** (NetWare 3.x and above). The difference between the NLM and the workstation is in the case of a local connection the NLM request goes through the NetWare API and directly into the NetWare OS. In the case of a remote connection, it goes out on the wire the same as a workstation request.

**Parent Topic:**

Connection Number and Task Management: Guides

# Overview of Connections and Tasks

A NetWare® server maintains a connection table which contains a series of slots that represent connection numbers. When a remote client logs in to a server, the server finds the first available slot and assigns that connection number to the client. That slot then becomes the remote client's connection number, which the client uses thereafter to identify itself to the server. The allocated connection numbers are not necessarily consecutive numbers, since a previously allocated connection number may be freed and available for another remote client.

Remote and Local Connections

Tasks in NetWare

NLM Applications and Connections

Current Connection and Task

**Parent Topic:**

Connection Number and Task Management: Guides

# Remote and Local Connections

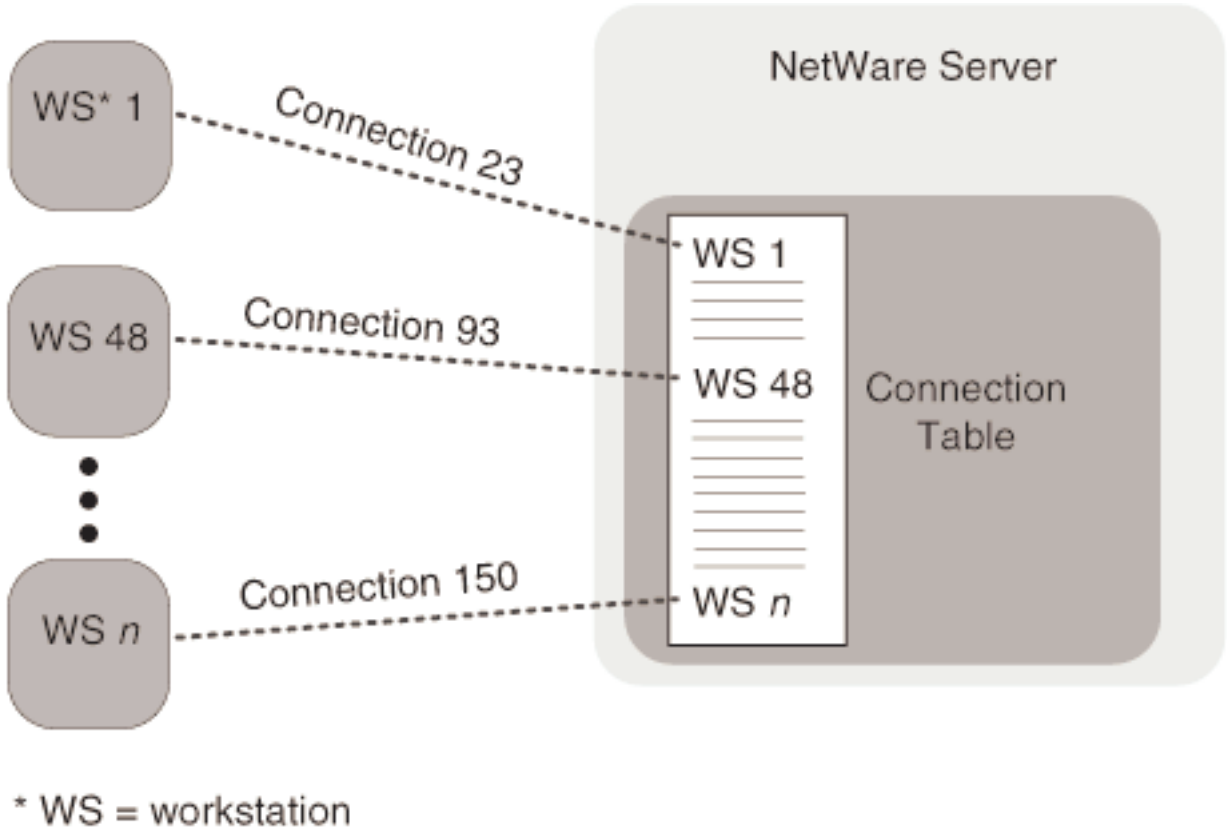
All connections are considered to be remote or local. A workstation connection is always a remote connection, since the machine it runs on is physically separate from the server. An NLM connection, on the other hand,

physically separate from the server. An NLM connection, on the other hand, can be remote or local. The connection is local when the NLM accesses the server the NLM is loaded on, but remote when accessing other servers.

By default, NLM applications are automatically allocated **connection zero (0)**. Connection 0 gives your NLM unlimited access to the local server's file system. In addition to connection 0, a local NLM frequently needs to get a connection to the local server, and it always needs to do so to gain access to a remote server.

The following figure shows a remote connection scenario. Workstations 1 and 48 have established connections to a server. Workstation 1 has connection number 23, workstation 48 has connection number 93, and so on. These workstations specify their connection numbers whenever they send a request to the server, which uses the number to verify security and carry out accounting and other functions.

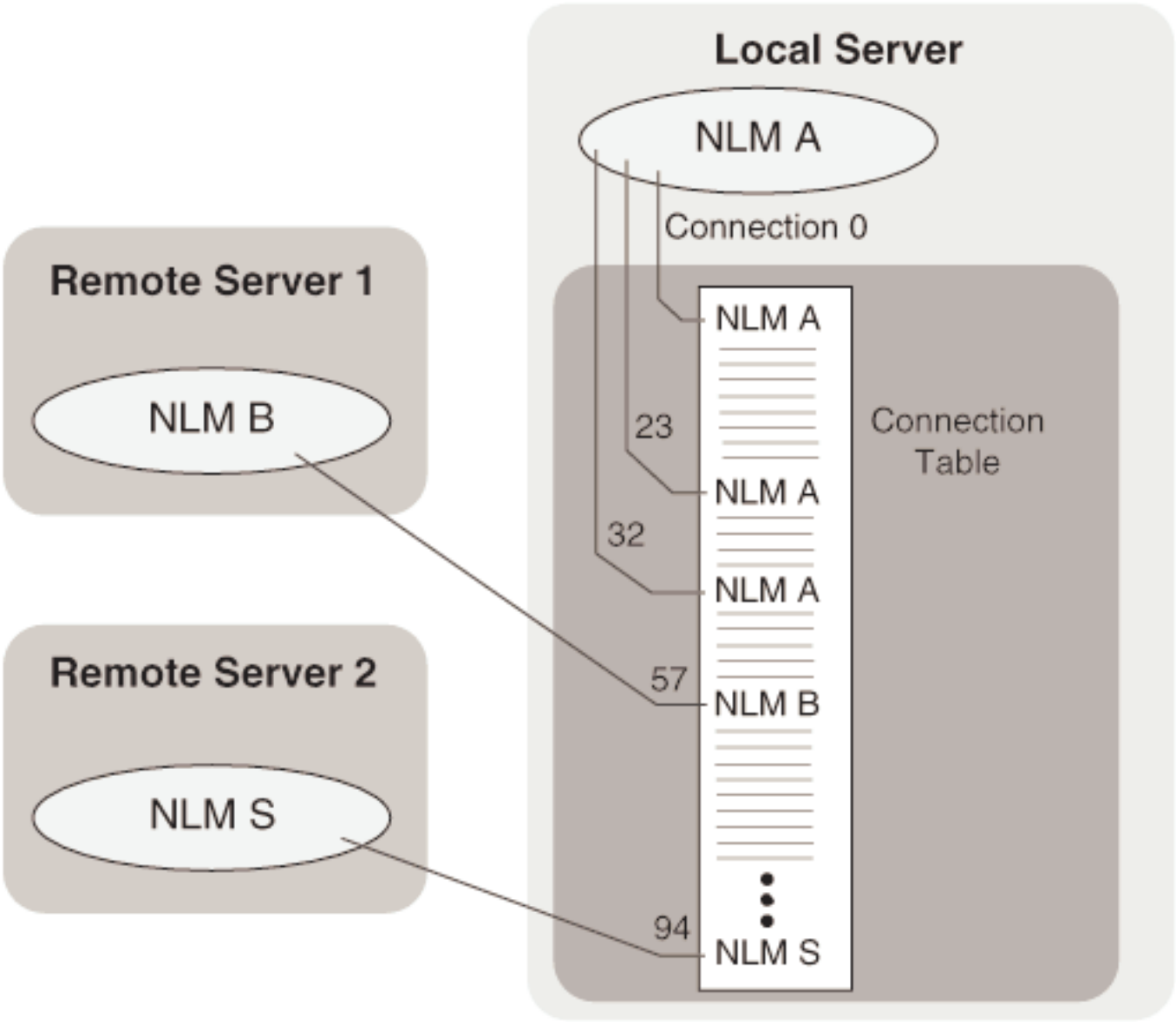
*Figure 1. Remote Connection to a Server*




The following figure shows NLM applications that have remote and local connections. NLM A is local to the server and has multiple connections including connection 0. NLM applications B and S have remote connections

to the server and therefore have connections other than 0.

*Figure 2. Remote and Local NLM Applications*




**Parent Topic:**

Connection Number and Task Management: Guides

## Single Connection, Many Users

An NLM can use task numbers to have the server perform tasks for multiple users on a single connection. For obvious reasons, each connection/task number pair you specify needs to be unique on that server. To allow your NLM to service multiple users on a single connection, allocate a block of tasks (**AllocateBlockOfTasks**) and then assign a different task to each user ( **SetCurrentTask**).

**Parent Topic:**

Connection Number and Task Management: Guides

# Tasks in NetWare

In NetWare, a task is a program running on a network workstation or server. NetWare uses a task number to identify each task.

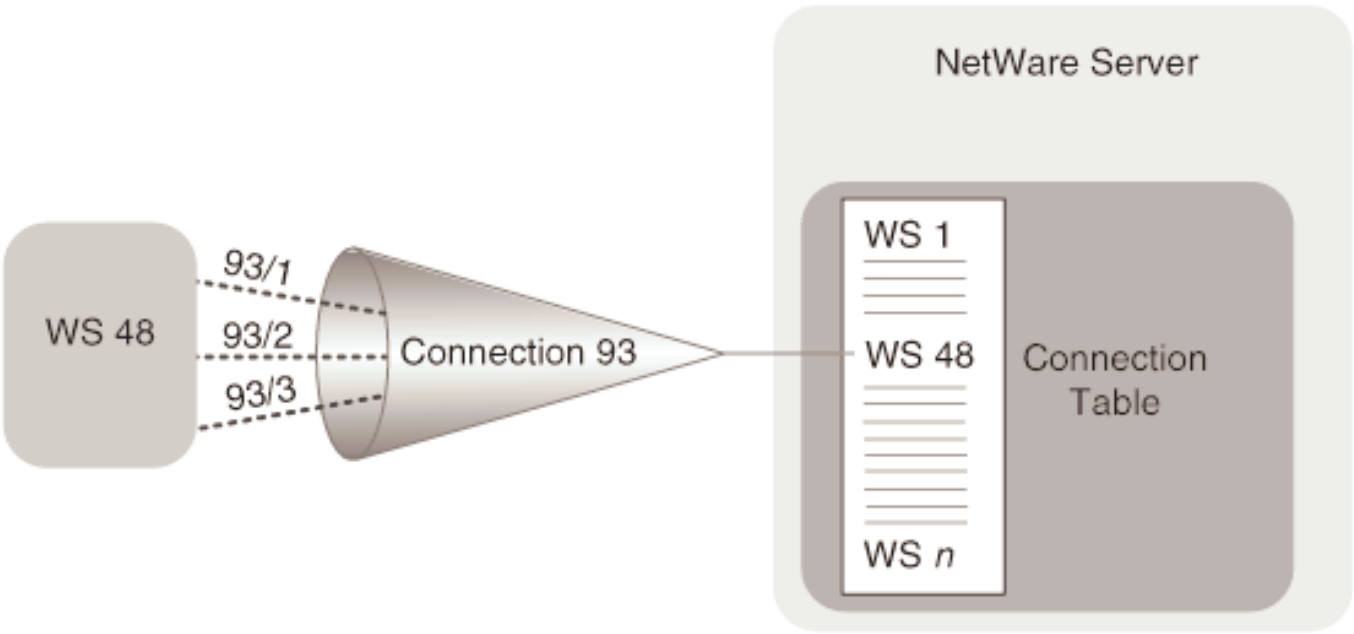
NetWare assigns task numbers sequentially, beginning with task number one. The combination of the connection number and the task number yields a unique connection/task number pair. This connection number/task number is unique only for a given computer.

NetWare uses the connection number/task number to manage network resources. Since NLM applications can access resources on their own behalf or on behalf of the client, NLM applications must specify both a connection number and a task number when making a request.

To run more than one session on a single connection, the server allocates task numbers, creating a connection/task number pair, as shown on Connection 93 in the following figure. Tasks 1, 2, and 3 are all being executed over Connection 93.

*Figure 3. Remote Connection Task Numbers*

*Connection Service Group*




**Parent Topic:**

Connection Number and Task Management: Guides

# Connection Number and Task Management: Functions

# AllocateBlockOfTasks

Returns a unique set of task numbers for the exclusive use of the requesting NLM™ application

**Local Servers:** nonblocking

**Remote Servers:** nonblocking

**Classification:** 3.x, 4.x

**SMP Aware:** No

**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

LONG AllocateBlockOfTasks (
   LONG   numberWanted);
```

## Parameters

*numberWanted*

(IN) Specifies the requested number of tasks in the set.

## Return Values

This function returns the first task number in the set. It returns a value of 0 if no task numbers are available.

## Remarks

Several entities can make requests to NetWare® 3.x and 4.x using a given connection number. Unique task numbers enable the calling NLM to use a connection number without any danger of conflict with other entities using that same connection. **AllocateBlockOfTasks** returns a unique set of task numbers, ensuring that the connection number/task number combination of the NLM is unique.

**AllocateBlockOfTasks** allocates the requested number of consecutive task numbers on the current connection for exclusive use by the NLM. An NLM that meets the following criteria would call this function:

Needs more than one task number for a given connection number.

Uses connection 0 or uses a client's connection number.

The **SetCurrentTask** function should be used if the NLM needs only one task.

### *See Also*

**ReturnBlockOfTasks, SetCurrentTask**

# CheckIfConnectionActive

Determines whether the specified connection number is processing a file-service request

**Local Servers:** nonblocking

**Remote Servers:** N/A

**Classification:** 3.x, 4.x

**SMP Aware:** No

**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

BYTE CheckIfConnectionActive (
   LONG   connectionNumber);
```

## Parameters

*connectionNumber*

(IN) Specifies the connection number being checked.

## Return Values

If the connection is active and processing a file service request, the function returns a value of 1. Otherwise, it returns a value of 0.

## Remarks

The **CheckIfConnectionActive** function determines whether the connection number is being used to process a file-service request. If the connection is being used for a service other than a file service request, this function returns 0.

## See Also

**DisableConnection**

# DisableConnection

Temporarily prevents the server from servicing any requests (except requests made by the calling NLM) for the specified connection number

**Local Servers:** nonblocking

**Remote Servers:** N/A

**Classification:** 3.x, 4.x

**SMP Aware:** Yes

**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

int DisableConnection (
   LONG   connection);
```

## Parameters

*connection*

(IN) Specifies the connection number to disable.

## Return Values

| | | | |
|---|---|---|---|
| 0 | (0x00) | ESUCCESS | Connection was disabled. |
| -1 | | EFAILURE | Connection is in use or already disabled. |

## Remarks

The **DisableConnection** function reserves the specified connection number solely for the use of the NLM. It prevents both the workstation and other NLM applications from using the connection number. However, other NLM applications must cooperate by also calling this function.

While the connection is temporarily disabled, the NLM can perform file service functions without any conflict from the workstation or other NLM applications. Disable a connection number for only a short period.

Once the NLM has used the specified connection number, the NLM should call **EnableConnection**, allowing the server to again service requests for the specified connection number.

An NLM should check the completion code of this function to make sure another NLM does not already have the connection disabled, or is otherwise in an incompatible state (such as disconnecting or processing an NCP request).

### See Also

**CheckIfConnectionActive**, **EnableConnection**

### Example

## DisableConnection

```
#include <stdlib.h>
#include <stddef.h>
#include <nwcntask.h>

main()
{
   LONG  connect;
   BYTE  buffer[100];

   printf("connection # ");
   gets(buffer);
   connect = atoi(buffer);
   printf("%d\r\n",DisableConnection(connect));
   getch();
   printf("%d\r\n",EnableConnection(connect));
   getch();
}
```

# EnableConnection

Enables the server to service requests for the specified connection
**Local Servers:** nonblocking
**Remote Servers:** N/A
**Classification:** 3.x, 4.x
**SMP Aware:** Yes
**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

int EnableConnection (
   LONG   connection);
```

## Parameters

*connection*

(IN) Specifies the connection number to enable.

## Return Values

| | | | |
|---|---|---|---|
| 0 | (0x00) | ESUCCESS | Connection was enabled. |

## Remarks

This function reverses the effect of the **DisableConnection** function, allowing the server to again service requests for the specified connection number.

Call **EnableConnection** only if you have previously called **DisableConnection** successfully.

## See Also

**DisableConnection**

## Example

### EnableConnection

```
#include <stdlib.h>
#include <stddef.h>
#include <nwcntask.h>

main()
{
   LONG    connect;
   BYTE    buffer[100];

   printf("connection # ");
   gets(buffer);
   connect = atoi(buffer);
   printf("%d\r\n",DisableConnection(connect));
   getch();
   printf("%d\r\n",EnableConnection(connect));
   getch();
}
```

# GetCurrentConnection

Returns the current connection on the current server
**Local Servers:** nonblocking
**Remote Servers:** nonblocking
**Classification:** 3.x, 4.x
**SMP Aware:** No
**Service:** Connection Number and Task Management

## *Syntax*

```
#include <nwcntask.h>

LONG GetCurrentConnection (void);
```

## *Return Values*

This function returns the current connection number.

## *Remarks*

The current connection number is returned for the calling thread group.

The **GetCurrentConnection** function is identical to **GetConnectionNumber**, except that the return type is LONG.

## *See Also*

**SetCurrentConnection**

# GetCurrentConnectionID

See **GetCurrentFileServerID**

# GetCurrentFileServerID

Returns the current file server ID number
**Local Servers:** nonblocking
**Remote Servers:** nonblocking
**Classification:** 3.11, 3.12, 4.x
**SMP Aware:** No
**Service:** Connection Number and Task Management

## *Syntax*

```
#include <nwcntask.h>

WORD GetCurrentFileServerID (void);
```

## *Return Values*

This function returns the current file server ID.

## *Remarks*

If the file server ID is nonzero, the current server is remote. If the file server ID is zero, the current server is local.

**GetCurrentFileServerID** is identical to **GetDefaultFileServerID**.

## *See Also*

**SetCurrentFileServerID**

# GetCurrentTask

Returns the calling thread group's current task number
**Local Servers:** nonblocking
**Remote Servers:** nonblocking
**Classification:** 3.x, 4.x
**SMP Aware:** No
**Service:** Connection Number and Task Management

## *Syntax*

```
#include <nwcntask.h>

LONG GetCurrentTask (void);
```

## *Return Values*

This function returns the current task number.

## *Remarks*

The current task number is returned for the calling thread group.

## *See Also*

**SetCurrentTask**

# LoginObject

Logs in the specified object to the specified connection number on the server
**Local Servers:** blocking
**Remote Servers:** blocking
**Classification:** 3.x, 4.x
**SMP Aware:** No
**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

int LoginObject (
   LONG   connection,
   char   *objectName,
   WORD   objectType,
   char   *password);
```

## Parameters

*connection*

(IN) Specifies the connection number the object is to be logged in to.

*objectName*

(IN) Specifies the string containing the name of the object.

*objectType*

(IN) Specifies the type of the object.

*password*

(IN) Specifies the string containing the object's password (lowercase passwords can be specified).

## Return Values

| | | |
|---|---|---|
| 0 | 0x 00 | ESUCCESS: Logout of object was successful |
| -1 | | EFAILURE: Connection number not valid |
| 150 | 0x 96 | ERR_SERVER_OUT_OF_MEMORY |

## Remarks

**LoginObject** logs in the object on the specified connection on the local server with the specified object name and type and with the specified password.

Call **SetCurrentConnection** first to select the connection, or call **GetCurrentConnection** to obtain the current connection. The current file server ID does not change.

The *objectType* parameter classifies an object as a user, user group, server, and so on. The following is a list of well-known object types:

| Value | Object Type |
|---|---|
| 0xFFFF | Wild |
| 0x0000 | Unknown |
| 0x0001 | User |
| 0x0002 | User Group |
| 0x0003 | Print Queue |
| 0x0004 | File Server |
| 0x0005 | Job Server |
| 0x0006 | Gateway |
| 0x0007 | Print Server |
| 0x0008 | Archive Queue |
| 0x0009 | Archive Server |
| 0x000A | Job Queue |
| 0x000B | Administration |
| 0x0024 | Remote Bridge Server |
| 0x0047 | Advertising Print Server |
| 0x004C | NetWare SQL |
| | |

| | |
|---|---|
| 0x8000 | Reserved up to |

An NLM typically uses object type 1 (user) or a type that the developer has requested from Novell®.

To log in as an object on a remote server, you must specify the object's password for the *password* parameter. On the local server, an NLM application can log in as an object without specifying the object's password. This is done by specifying LOGIN_WITHOUT_PASSWORD for the *password* parameter. If your application uses LOGIN_WITHOUT_PASSWORD, it must ensure that NetWare security is not breached.

### See Also

**NWLoginToFileServer, LogoutObject**

### Example

## LoginObject

```
#include   <stdio.h>
#include   <stdlib.h>
#include   <nwcntask.h>
#include   <nwbindry.h>

main()
{
   BYTE   *name;
   LONG   log;
   name = "supervisor";
   printf ("SetCurrentConnection = %d\r\n", SetCurrentConnection(-1));
   log = LoginObject (GetCurrentConnection(), name, 0T_USER, " ");
   if(log)
   {
      printf ("login failed, rc = %d\n", log);
      getch ();
      return 1;
   }
   getch ();
   printf ("LogoutObject return = %d\r\n",
      LogoutObject (GetCurrentConnection() ));
}
```

# LogoutObject

Logs out the logged-in object on the specified connection on the current server

**Local Servers:** blocking

**Remote Servers:** blocking

**Classification:** 3.x, 4.x

**SMP Aware:** No

**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

int LogoutObject (
   LONG   connection);
```

## Parameters

*connection*

(IN) Specifies the connection number from which the object is to be logged out.

## Return Values

| | | | |
|---|---|---|---|
| 0 | (0x00) | ESUCCESS | Logout of object was successful. |
| -1 | | EFAILURE | Invalid connection number or NLM has not logged in to the specified connection. |
| 150 | (0x96) | ERR_SERVER_OUT_OF_MEMORY | |

## Remarks

**LogoutObject** only logs out connections on the current server (currently selected file server ID).

**LogoutObject** destroys your connection to a remote server. Therefore, if your current connection is to that server, your current connection is changed. On a local server, this function does not destroy your connection.

*Connection Service Group*

### *See Also*

**LoginObject**

# ReturnBlockOfTasks

Frees a block of task numbers
**Local Servers:** blocking
**Remote Servers:** nonblocking
**Classification:** 3.x, 4.x
**SMP Aware:** No
**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

int ReturnBlockOfTasks (
   LONG   startingTask,
   LONG   numberOfTasks);
```

## Parameters

*startingTask*

(IN) Specifies the first task number in the set.

*numberOfTasks*

(IN) Specifies the number of task numbers in the set.

## Return Values

| | | | |
|---|---|---|---|
| 0 | (0x00) | ESUCCESS | Block of task numbers are freed. |
| NetWare Error | | | Not successful. |

## Remarks

The **ReturnBlockOfTasks** function frees one or more task numbers the NLM previously allocated with the **SetCurrentTask** or **AllocateBlockOfTasks** function. An NLM should call the **ReturnBlockOfTasks** function to return the allocated task numbers before unloading.

## See Also

**AllocateBlockOfTasks**, **SetCurrentTask**

# ReturnConnection

Returns a connection number the NLM previously allocated
**Local Servers:** blocking
**Remote Servers:** blocking
**Classification:** 3.x, 4.x
**SMP Aware:** No
**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

int ReturnConnection (
   LONG   connectionNumber);
```

## Parameters

*connectionNumber*

(IN) Specifies the connection number obtained with **SetCurrentConnection**.

## Return Values

| | | | |
|---|---|---|---|
| 0 | (0x00) | ESUCCESS | Connection number returned. |

## Remarks

**ReturnConnection** returns a connection number previously allocated by **SetCurrentConnection**, **LoginObject**, **NWAttachToFileServer**, or **LoginToFileServer**.

## See Also

**NWAttachToFileServer**, **LoginObject**, **NWLoginToFileServer**, **SetCurrentConnection**

## Example

### ReturnConnection

```
#include <errno.h>
#include <nwcntask.h>
```

```
#include "mystuff.h"

main()
{
   LONG   conn;
   conn = SetCurrentConnection (-1);
   if (conn == EFAILURE) return 1;
   rc = LoginObject (conn, "serverX", MY_TYPE," ");
   if (rc) return 2;
   rc = LogoutObject (conn);
   if (rc) return 3;
   return ReturnConnection (conn);
}
```

# SetCurrentConnection

Changes the current connection number for the current thread group or allocates a new connection number

**Local Servers:** nonblocking

**Remote Servers:** blocking

**Classification:** 3.x, 4.x

**SMP Aware:** No

**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

LONG SetCurrentConnection (
   LONG   connectionNumber);
```

## Parameters

*connectionNumber*

(IN) Specifies the connection number to set.

## Return Values

If successful, this function returns the connection number that was current when you changed it so that you can change back to it if you want to. If not successful, this function returns EFAILURE (-1).

## Remarks

The **SetCurrentConnection** function sets the current connection number for the current thread group. You can either pass -1 or a connection number. If you pass -1, you allocate a new connection number for the exclusive use of your NLM, which is made the thread group's current connection. If you pass any other value, you change the thread group's current connection to that value.

For example, if you have four connections, 1 through 4, and the current connection is 2, you can change the current connection to 4 by passing 4. If you get back 2, you know you have successfully changed the current connection to 4. On the other hand, if you want to allocate a new connection, you pass -1. If successful, you still get back 2, and your current connection is an unknown number, which you can identify by calling **GetCurrentConnection** (or by calling **SetCurrentConnection** again to see what it returns).

When setting connections on the local server, you can set any available

connection number; however, when setting connection numbers on a remote server, you can set only those that your NLM has logged in on.

### See Also

**GetCurrentConnection**, **ReturnConnection**

### Example

## SetCurrentConnection

```
#include <stdio.h>
#include <stdlib.h>
#include <nwcntask.h>
#include <errno.h>

main()
{
   int   rc;
   rc = SetCurrentConnection (-1);
   printf ("SetCurrentConnection rc = %d\n", rc);
   if (rc == EFAILURE) return 1;
   printf("SetCurrentTask return value:%d\r\n",
           SetCurrentTask(5));
   printf("GetCurrentConnection %d\r\n",
           GetCurrentConnection());
   printf("GetCurrentTask (should be 5):%d\r\n",
           GetCurrentTask());
   getch();
}
```

# SetCurrentConnectionID

See **SetCurrentFileServerID**

# SetCurrentFileServerID

Sets the current connection ID (file server ID)
**Local Servers:** N/A
**Remote Servers:** blocking
**Classification:** 3.x, 4.x
**SMP Aware:** No
**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

WORD SetCurrentFileServerID (
   WORD   fileServerID);
```

## Parameters

*fileServerID*

(IN) Specifies the file server ID to set.

## Return Values

This function returns the old file server ID if successful. Otherwise, it returns EFAILURE.

## Remarks

If the file server ID is nonzero (remote server), a login operation must have previously been performed on that server or **SetCurrentFileServerID** returns an error.

After calling **SetCurrentFileServerID**, call **SetCurrentConnection** to set the correct connection.

When changing to a remote server, the current connection is set to the first connection number in the connection list for that server. **SetCurrentConnection** can then be used to specify some other connection.

When changing to a local server (zero), the current connection is set to the first connection number in the local connection list. If no local logins have been performed, the current connection is set to zero.

## See Also

**GetCurrentFileServerID**

# SetCurrentTask

Sets the calling thread group's current task number
**Local Servers:** nonblocking
**Remote Servers:** nonblocking
**Classification:** 3.x, 4.x
**SMP Aware:** No
**Service:** Connection Number and Task Management

## Syntax

```
#include <nwcntask.h>

LONG SetCurrentTask (
   LONG   taskNumber);
```

## Parameters

*taskNumber*

(IN) Specifies the task number to set.

## Return Values

This function returns the old current task number if successful. Otherwise, it returns EFAILURE.

## Remarks

This function sets the current task number for the thread group. If the *taskNumber* parameter is -1, a new task number is allocated. If the *taskNumber* parameter is not -1, the current task is set to that value.

Call **SetCurrentTask** if the NLM needs to allocate only one task number for the current connection number. If more than one task number is needed, call **AllocateBlockOfTasks**.

## See Also

**AllocateBlockOfTasks**, **GetCurrentTask**, **ReturnBlockOfTasks**