Novell Developer Kit

www.novell.com

March 1, 2006

CONNECTION, MESSAGE, AND NCP™ EXTENSIONS

Novell®

Legal Notices

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

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

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

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

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

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

Novell Trademarks

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

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

TTS is a trademark of Novell, Inc.

Universal Component System is a registered trademark of Novell, Inc.

Virtual Loadable Module is a trademark of Novell, Inc.

VLM is a trademark of Novell, Inc.

Yes Certified is a trademark of Novell, Inc.

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

Third-Party Materials

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

Contents

About This Guide

This guide provides information about two kinds of connection services:

- Cross-platform service simply referred to as a connection

  This service provides for attached, authenticated, and licensed connection states, and uses a connection reference as the caller's immediate access to the connection.

- NLM specific service called connection number and task management.

  The connections in this service are more directly tied to a specific server, and use connection numbers that are directly associated with specific slots on the server's connection table. The model is well designed for NLMs that require direct access to the server. However, NLMs themselves can also act as clients and use cross-platform connections, provided the NLMs are properly written for that purpose.

In addition, this guide contains information about the Message functions, which allow your application to send broadcast messages to other workstations attached to a common NetWare server, and the NCP Extension functions, which allow you to extend the services provided by the NetWare OS while maintaining the advantages associated with NCPs.

This guide includes the following functions:

Feedback

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

Documentation Updates

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

Additional Information

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

- *NDK: NLM Development Concepts, Tools, and Functions*
- *NDK: Program Management*
- *NDK: NLM Threads Management*
- *NDK: Multiple and Inter-File Services*

- *NDK: Single and Intra-File Services*
- *NDK: Volume Management*
- *NDK: Client Management*
- *NDK: Network Management*
- *NDK: Server Management*
- *NDK: Internationalization*
- *NDK: Unicode*
- *NDK: Sample Code*
- *NDK: Getting Started with NetWare Cross-Platform Libraries for C*
- *NDK: Bindery Management*

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

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

Documentation Conventions

In this documentation, a greater-than symbol (>) is used to separate actions within a step and items within a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

Connection Concepts

1

This documentation describes Connection, its functions, and features.

1.1 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. NDS 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.

1.2 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.

1.3 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 (page 69). The reference cannot be used in place of the connection handle. However, by calling NWCCOpenConnByRef (page 60), 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 (page 37).

1.4 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 NDS.

1.5 Open and Close Functions

These functions open and close a connection:

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.

Function	Description
NWCCSysCloseConnRef	Closes and detaches the specified connection, including the connection reference and all connection handles for this connection.
NWCCUnlicenseConn	Unlicenses the specified licensed connection.

1.6 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.
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.

1.7 Get Information Functions

These functions get information about a connection:

Function	Description
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.
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.

Function	Description
NWCCScanConnRefs	Returns a connection reference for each connection on the workstation.

1.8 Set Parameter Functions

These functions set connection parameters:

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.

Connection Tasks

2

This documentation describes common tasks associated with Connection.

2.1 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 (see Section 5.7, “NWCC_INFO_LICENSE_STATE Values,” on page 137).

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.

See openconn.c (../../../samplecode/clib_sample/connect/openconn/openconn.c.html) for sample code.

2.2 Getting Connection Status

The following functions get server connection information:

- NWCCGetAllConnInfo (page 26)
- NWCCGetConnAddress (page 31)
- NWCCGetConnRefInfo (page 42)
- NWCCGetAllConnRefInfo (page 28)
- NWCCGetConnRefAddress (page 38)

All existing workstation connections can be scanned, including matching information for all existing connections. This is done using NWCCScanConnRefs (page 69).

The following functions retrieve connection information:

- NWCCGetPrefServerName (page 45)
- NWCCGetPrimConnRef (page 46)

2.3 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 (page 49)
- NWCCUnlicenseConn (page 79)

  The connection is unlicensed only if there are no open handles that require a licensed connection. See licconn.c (../../../samplecode/clib_sample/connect/licconn/licconn.c.html) for sample code.

The following functions set connections to special cases:

- NWCCMakeConnPermanent (page 51)

  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 (page 77).
- NWCCSetPrefServerName (page 73)
- NWCCSetPrimConn (page 74)

2.4 Closing and Clearing Connections

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.

A connection handle can be closed by calling NWCCCloseConn (page 24).

A connection reference is cleared by following these steps:

1. Call NWCCGetConnRef (page 37) to obtain the connection reference to clear.
2. Pass the connection reference returned from NWCCGetConnRef (page 37) to NWCCSysCloseConnRef (page 77) which will clear the specified connection reference.

NWCCSysCloseConnRef (page 77) forces a server connection detach (used in conjunction with NWCCMakeConnPermanent (page 51)).

IMPORTANT: Use caution when calling NWCCSysCloseConnRef (page 77) because other applications on the workstation might be using the connection.

2.5 Listing Connection Handles

You can obtain a list of allocated connection handles by calling NWCCScanConnRefs (page 69) followed by NWCCOpenConnByRef (page 60). 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 (page 44) to find the maximum number of connections supported.

Other connection handle functions include the following:

- NWCCScanConnRefs (page 69) followed by the NWCCOpenConnByRef (page 60)

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

2.6 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:

- NWClearConnectionNumber (page 81)
- NWGetObjectConnectionNumbers (page 107)

Connection Functions

3

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

IMPORTANT: You can establish a TCP or UDP connection only on a NetWare 5.x or 6.x server.

3.1 NWCCA*-NWCCK* Functions

Click on any function name in the table of contents to view the purpose, syntax, parameters, and return values for that function.

NWCCCloseConn

Closes the specified connection

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCCloseConn  (
   NWCONN_HANDLE   connHandle);
```

Delphi

```
uses clxwin32

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

Parameters

connHandle

(IN) Specifies the connection handle to be closed.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID
0x8869	NWE_ACCESS_VIOLATION
0x886C	NWE_RESOURCE_LOCK
0x8872	NWE_INVALID_OWNER
0x890A	NLM_INVALID_CONNECTION

Remarks

NWCCCloseConn is used to close an open connection handle. Calling NWCCCloseConn has the opposite effect as the following 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 server connection table.

See Also

NWCCOpenConnByName (page 55), NWCCOpenConnByPref (page 58), NWCCOpenConnByRef (page 60), NWClearConnectionNumber (page 81)

NWCCGetAllConnInfo

Returns all information for the specified connection

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID

0x8868	NWE_STRING_TRANSLATION
0x8869	NWE_ACCESS_VIOLATION
0x890A	NLM_INVALID_CONNECTION

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.

See Also

NWGetConnectionInformation (page 91), NWGetUserInfo (Server Management)

NWCCGetAllConnRefInfo

Returns all information for a specified connection reference

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID

0x8868	NWE_STRING_TRANSLATION
0x8869	NWE_ACCESS_VIOLATION
0x890A	NLM_INVALID_CONNECTION

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.

See Also

NWGetConnectionInformation (page 91), NWGetUserInfo (Server Management)

NWCCGetCLXVersion

Returns the version of the current CLX layer

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

Procedure NWCCGetCLXVersion
  (Var majorVersion : nuint8;
   Var minorVersion : nuint8;
   Var revisionLevel : nuint8;
   Var 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.

NWCCGetConnAddress

Returns the transport address for the specified connection

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 user allocated buffer pointed to by the `buffer` field in the NWCCTranAddr structure (see "Remarks" below).

tranAddr

(IN/OUT) Points to an NWCCTranAddr structure that contains the type and length of the transport address and points to a user allocated buffer for receiving the address on return.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESS

0x8801	NWE_CONN_INVALID
0x8867	NWE_INSUFFICIENT_RESOURCES
0x8868	NWE_STRING_TRANSLATION
0x8869	NWE_ACCESS_VIOLATION
0x890A	NLM_INVALID_CONNECTION

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.

NWCCGetConnAddressLength

Returns the length of the connection address for the specified connection

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID
0x8868	NWE_STRING_TRANSLATION
0x8869	NWE_ACCESS_VIOLATION
0x890A	NLM_INVALID_CONNECTION

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.

NWCCGetConnInfo

Returns information about the specified connection

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (see Section 5.4, “infoType Parameter Values,” on page 136). *NOTE: Do not pass NWCC_INFO_RETURN_ALL (see "Remarks" below).*

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID
0x8868	NWE_STRING_TRANSLATION
0x8869	NWE_ACCESS_VIOLATION
0x886B	NWE_INVALID_LEVEL
0x890A	NLM_INVALID_CONNECTION

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 Section 5.4, “infoType Parameter Values,” on page 136).

NOTE: Do not pass NWCC_INFO_RETURN_ALL for the `infoType` parameter. NWCCGetConnInfo fails on that value and returns NWE_INVALID_LEVEL. To obtain all information about a connection, call NWCCGetAllConnInfo (page 26).

See

- Section 5.5, “NWCC_INFO_AUTHENT_STATE Values,” on page 137
- Section 5.6, “NWCC_INFO_BCAST_STATE Values,” on page 137
- Section 5.7, “NWCC_INFO_LICENSE_STATE Values,” on page 137
- Section 5.8, “NWCC_INFO_NDS_STATE Values,” on page 138.

NWCCGetConnRef

Returns the connection reference for the specified connection

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESS

NWCCGetConnRefAddress

Returns the transport address for the specified connection reference

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (*Return Values for C*) 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
0x890A	NLM_INVALID_CONNECTION

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.

NWCCGetConnRefAddressLength

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

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

Function NWCCGetConnRefAddressLength
  (connRef : nuint32;
   Var 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 (*Return Values for C*) 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
0x890A	NLM_INVALID_CONNECTION

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 Section 5.4, “infoType Parameter Values,” on page 136.

NWCCGetConnRefInfo

Returns the specified information for a given connection reference

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (see Section 5.4, “infoType Parameter Values,” on page 136).

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 (*Return Values for C*) 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
0x890A	NLM_INVALID_CONNECTION

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 Section 5.4, “infoType Parameter Values,” on page 136.

If the `infoType` parameter is set to NWCC_INFO_TRAN_ADDR, NWCCGetConnRefInfo will use the NWCCTranAddr (page 129) structure.

See

- Section 5.5, “NWCC_INFO_AUTHENT_STATE Values,” on page 137
- Section 5.6, “NWCC_INFO_BCAST_STATE Values,” on page 137
- Section 5.7, “NWCC_INFO_LICENSE_STATE Values,” on page 137
- Section 5.8, “NWCC_INFO_NDS_STATE Values,” on page 138.

NWCCGetNumConns

Returns the current and maximum number of connections for the requester

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

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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).

NWCCGetPrefServerName

Returns the name from the PREFERRED SERVER parameter

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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.

NWCCGetPrimConnRef

Returns the primary connection reference

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

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCGetPrimConnRef (
   pnuint32   connRef);
```

Delphi

```
uses clxwin32

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 (*Return Values for C*) 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.

If NWCCGetPrimConnRef is called on the NLM platform, NWE_FUNCTION_INVALID will be returned.

NWCCGetSecurityFlags

Returns the configured security flags for the requester

NetWare Server: 4.1x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (seeSection 5.11, “Security Flag Values,” on page 139).

prefSecurityFlags

(OUT) Points to the preferred (but not required) security level (seeSection 5.11, “Security Flag Values,” on page 139).

reqSecurityFlags

(OUT) Points to the required security flags for each connection (see Section 5.11, “Security Flag Values,” on page 139).

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESS

Remarks

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.

See Also

NWCCSetSecurityFlags (page 75)

3.2 NWCCL*-NWCCZ* Functions

Click on any function name in the table of contents to view the purpose, syntax, parameters, and return values for that function.

NWCCLicenseConn

Licenses the specified connection

NetWare Server: 3.x, 4.x, 5.x, 6.x

Platform: NLM, Windows NT

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCLicenseConn (
   NWCONN_HANDLE   connHandle);
```

Delphi

```
uses clxwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID
0x8869	NWE_ACCESS_VIOLATION
0x8872	NWE_INVALID_OWNER
0x890A	NLM_INVALID_CONNECTION

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.

Windows NT will return SUCCESS if the specified handle is already licensed.

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

NWCCLicenseConn is supported under VLMs but not supported on client32 requesters.

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

See licconn.c (../../../samplecode/clib_sample/connect/licconn/licconn.c.html) for sample code.

NWCCMakeConnPermanent

Keeps the specified connection from being detached until NWCCSysCloseConnRef is called

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCMakeConnPermanent (
   NWCONN_HANDLE   connHandle);
```

Delphi

```
uses clxwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID
0x8869	NWE_ACCESS_VIOLATION
0x8872	NWE_INVALID_OWNER
0x890A	NLM_INVALID_CONNECTION

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.

NWCCOpenConnByAddr

Opens a connection using a network address

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCOpenConnByAddr (
   const NWCCTranAddr N_FAR  *tranAddr,
   nuint                      openState,
   nuint                      reserved,
   pNWCONN_HANDLE             pConnHandle);
```

Delphi

```
uses clxwin32

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 (NWCC_TRAN_TYPE_WILD does not apply to NWCCOpenConnByAddr).

openState

(IN) Specifies the state of the connection (see Section 5.2, “Connection State Values,” on page 135).

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 (*Return Values for C*) 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

NWCCOpenConnByName

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

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCOpenConnByName (
   NWCONN_HANDLE       startConnHandle,
   const nstr8 N_FAR  *name,
   nuint               nameFormat,
   nuint               openState,
   nuint               tranType,
   pNWCONN_HANDLE      pConnHandle);
```

Delphi

```
uses clxwin32

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

Parameters

startConnHandle

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

name

(IN) Points to the name of the server or tree to connect to.

nameFormat

(IN) Specifies the format of the server or tree to connect to (see Section 5.9, “Name Format Values,” on page 138).

openState

(IN) Specifies the desired open state of the connection (see Section 5.2, “Connection State Values,” on page 135).

tranType

(IN) Specifies the transport type (see Section 5.12, “Transport Type Values,” on page 139) and "Remarks" below.

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 (*Return Values for C*) 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
0x8904	The server name is invalid.
0x890A	NLM_INVALID_CONNECTION
0xAF7E	The tree and server name combination is invalid.

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 do not care which connection is used to resolve the name.

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

For the `tranType` parameter, Novell recommends passing NWCC_TRAN_TYPE_WILD and letting the underlying system use the type best suited for the connection it is making. The other type values remain available, however.

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.

See openconn.c (../../../samplecode/clib_sample/connect/openconn/openconn.c.html) for sample code.

See Also

NWCCCloseConn (page 24), NWCCOpenConnByPref (page 58), NWCCOpenConnByRef (page 60)

NWCCOpenConnByPref

Opens an initial connection using the configured preferred settings

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

Function NWCCOpenConnByPref (
   tranType : nuint;
   openState : nuint;
   reserved : nuint;
   pConnHandle : pNWCONN_HANDLE
) : NWRCODE;
```

Parameters

tranType

(IN) Specifies the preferred or required transport type to be used (see Section 5.12, “Transport Type Values,” on page 139).

openState

(IN) Specifies the state of the connection (see Section 5.2, “Connection State Values,” on page 135).

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 (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	NWE_CONN_INVALID
0x8808	NWE_SERVER_NO_SLOTS
0x880A	NWE_SERVER_NO_ROUTE
0x883F	NWE_CONN_TABLE_FULL
0x8847	NWE_SERVER_NOT_FOUND
0x8867	NWE_INSUFFICIENT_RESOURCES
0x8869	NWE_ACCESS_VIOLATION
0x8870	NWE_UNSUPPORTED_TRAN_TYPE
0x890A	NLM_INVALID_CONNECTION

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.

See Also

NWCCCloseConn (page 24), NWCCOpenConnByName (page 55), NWCCOpenConnByRef (page 60)

NWCCOpenConnByRef

Opens a connection associated with the given connection reference

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (see Section 5.2, “Connection State Values,” on page 135).

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID
0x8836	INVALID_PARAMETER
0x8869	NWE_ACCESS_VIOLATION
0x890A	NLM_INVALID_CONNECTION

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.

See Also

NWCCCloseConn (page 24), NWCCOpenConnByName (page 55), NWCCOpenConnByPref (page 58)

NWCCQueryFeature

Determines if the Requester supports a given feature

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCQueryFeature (
   nuint   featureCode);
```

Delphi

```
Function NWCCQueryFeature (
   featureCode : nuint
) : NWRCODE;
```

Parameters

`featureCode`

(IN) Specifies the feature being queried (see Section 5.3, “Feature Code Values,” on page 136).

Return Values

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

NWCCRenegotiateSecurityLevel

Sets a new security level for the specified connection

NetWare Server: 4.1x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (see Section 5.11, "Security Flag Values," on page 139).

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

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

0x890A	NLM_INVALID_CONNECTION

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.

See Also

NWCCOpenConnByAddr (page 53), NWCCOpenConnByName (page 55), NWCCOpenConnByPref (page 58), NWCCOpenConnByRef (page 60), NWCCSysCloseConnRef (page 77)

NWCCRequest

Sends a fragment-based NCP request directly to the specified server.

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

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

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

NWRCODE NWCCRequest (
   NWCONN_HANDLE           connHandle,
   nuint                   function,
   nuint                   numReqFrags,
   const NWCCFRAG N_FAR   *reqFrags,
   nuint                   numReplyFrags,
   pNWCCFRAG               replyFrags,
   pnuint                  actualReplyLen);
```

Parameters

connHandle

(IN) Specifies the NetWare server connection handle where the request is being directed.

function

(IN) Specifies the NCP function number being requested.

numReqFrags

(IN) Specifies the number of fragments that make up the request packet pointed to by `reqFrags` (maximum is five).

reqFrags

(IN) Points to the array of fragments that make up the request packet.

numReplyFrags

(IN) Specifies the number of fragments that make up the reply packet pointed to by `replyFrags` (maximum is five).

replyFrags

(OUT) Points to the array of fragments that make up the reply packet.

actualReplyLen

(OUT) Points to the total size of the reply packet as described in the NCP header after any heading or trailing information is removed (optional).

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	NWE_CONN_INVALID: Bad connection handle
0x8869	NWE_ACCESS_VIOLATION: Connection handle belongs to another process
0x89XX	Server error from NCP or Requester

Remarks

NWCCRequest is the simplest and most common method for sending NCP requests to a server.

`reqFrags` and `replyFrags` can be NULL if the corresponding `numReqFrags` and `numReplyFrag` parameter is zero.

The request and reply fragment buffers should not overlap.

See Also

NWRequest (page 111)

NWCCScanConnInfo

Returns connection information for multiple connections

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

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCScanConnInfo (
   pnuint32           scanIterator,
   nuint              scanInfoLevel,
   const void N_FAR  *scanConnInfo,
   nuint              scanFlags,
   nuint              connInfoVersion,
   nuint              returnInfoLevel,
   nptr               returnConnInfo,
   pnuint32           connReference);
```

Delphi

```
uses clxwin32

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

Parameters

`scanIterator`

(IN/OUT) Specifies 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 (see Section 5.4, “infoType Parameter Values,” on page 136).

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 (see Section 5.10, "Scan Flag Values," on page 138).

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 (see Section 5.4, "infoType Parameter Values," on page 136).

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 (*Return Values for C*) 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

Either the entire NWCCConnInfo (page 125) structure or part of the NWCCConnInfo (page 125) structure can be returned using the `returnConnInfo` parameter. If the entire NWCCConnInfo (page 125) structure is being returned (`returnInfoLevel` =NWCC_INFO_RETURN_ALL), the `returnConnInfo` parameter must point to a buffer of type NWCCConnInfo (page 125). If part of the NWCCConnInfo (page 125) 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 (page 129) structure was passed an incorrect amount. The NWCCGetConnRefInfo function can be called to retrieve the transport address.

NWCCScanConnRefs

Returns a connection reference for each connection on the workstation

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (*Return Values for C*) 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.

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, 5.x, 6.x

Platform: NLM

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCSetCurrentConnection  (
   CONN_HANDLE   connHandle);
```

Delphi

```
uses clxwin32

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

Parameters

connHandle

(IN) Specifies the connection handle.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESS
0x890A	INVALID_CONNECTION

Remarks

NWCCSetCurrentConnection is similar to SetCurrentConnection, 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.

See Also

fopen(Single and Intra-File Services), NWCCOpenConnByName (page 55), SetCurrentConnection (page 166)

NWCCSetPrefServerName

Sets the `prefServer` parameter for the workstation

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCSetPrefServerName (
   const nstr N_FAR  *prefServer);
```

Delphi

```
uses clxwin32

Function NWCCSetPrefServerName
  (const 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 (*Return Values for C*) 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.

NWCCSetPrimConn

Sets the primary connection for the workstation

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCSetPrimConn  (
   NWCONN_HANDLE   connHandle);
```

Delphi

```
uses clxwin32

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 (*Return Values for C*) 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.

NWCCSetSecurityFlags

Sets the configured security flags for the requester

NetWare Server: 4.1x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8836	NWE_PARAM_INVALID
0x8861	NWE_SIGNATURE_LEVEL_CONFLICT

Remarks

See Section 5.11, “Security Flag Values,” on page 139.

See Also

NWCCGetSecurityFlags (page 47)

NWCCSysCloseConnRef

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

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCSysCloseConnRef  (
   nuint32   connRef);
```

Delphi

```
uses clxwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID
0x8869	NWE_ACCESS_VIOLATION
0x890A	NLM_INVALID_CONNECTION

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.

NWCCUnlicenseConn

Unlicenses the specified licensed connection

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

N_EXTERN_LIBRARY NWRCODE NWCCUnlicenseConn  (
   NWCONN_HANDLE   connHandle);
```

Delphi

```
uses clxwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESS
0x8801	NWE_CONN_INVALID
0x8815	NWE_HANDLE_ALREADY_UNLICENSED
0x8869	NWE_ACCESS_VIOLATION
0x8872	NWE_INVALID_OWNER
0x890A	NLM_INVALID_CONNECTION

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.

See licconn.c (../../../samplecode/clib_sample/connect/licconn/licconn.c.html) for sample code.

3.3 NWCI*-NWGetH* Functions

Click on any function name in the table of contents to view the purpose, syntax, parameters, and return values for that function.

NWClearConnectionNumber

Logs out the specified connection

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Connection

Syntax

C

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

NWCCODE N_API  NWClearConnectionNumber  (
   NWCONN_HANDLE   connHandle,
   nuint16         connNumber);
```

Delphi

```
uses calwin32

Function NWClearConnectionNumber
  (connHandle : NWCONN_HANDLE;
   connNumber : nuint16
) : 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 (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

0x890A	NLM_INVALID_CONNECTION
0x89C6	NO_CONSOLE_PRIVILEGES
0x89FD	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 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 (page 24)

NWCLXInit

Initializes the CLX (connection API) library

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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

Parameters

reserved1

(IN) Is reserved for future use (pass in NULL).

reserved2

(IN) Is reserved for future use (pass in NULL).

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL

Remarks

NWCLXInit initializes only the CLX (connection API) libraries. If you are using only the connection API functions (NWCC . . .), you can initialize the libraries by calling NWCLXInit. You do not need to call NWCallsInit for these libraries.

Novell recommends you call NWCLXInit before calling any NWCC. . . function.

When you have finished using the CLX libraries after calling NWCLXInit, call NWCLXTerm (page 85) to terminate the libraries and perform clean up.

See Also

NWCLXTerm (page 85)

NWCLXTerm

Terminates the CLX library and performs any necessary clean up

NetWare Server: 3.x, 4.x, 5.x, 6.x

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

Library: Cross-Platform Client (CLX*.*)

Service: Connection

Syntax

C

```
#include <nwclxcon.h>

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

Delphi

```
uses clxwin32

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

Parameters

reserved

(IN) Is reserved for future use {pass in NULL).

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL

Remarks

IMPORTANT: Call NWCLXTerm only if you have previously called NWCLXInit.

See Also

NWCLXInit (page 83)

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, 5.x, 6.x

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

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

Service: Connection

Syntax

C

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

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

Delphi

```
uses netwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESSFUL

0x8801	INVALID_CONNECTION
0x8836	INVALID_PARAMETER
0x890A	NLM_INVALID_CONNECTION
0xFEA0	ERR_NO_WRITABLE_REPLICAS
0xFEB8	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 (page 53), NWCCScanConnRefs (page 69), NWCCOpenConnByRef (page 60), NWCCLicenseConn (page 49)

NWGetConnectionIDFromAddress (obsolete-moved from .h file 11/99)

Was last documented in September 1999. Call the NWCCScanConnInfo (page 67), NWCCOpenConnByRef (page 60), and NWCCLicenseConn (page 49) functions instead.

NWGetConnectionIDFromName (obsolete-moved from .h file 11/99)

Was last documented in September 1999. Call the NWCCScanConnInfo (page 67), NWCCOpenConnByRef (page 60), and NWCCLicenseConn (page 49) functions instead.

NWGetConnectionInformation

Returns information about a logged in object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x

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

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

Service: Connection

Syntax

C

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

NWCCODE N_API  NWGetConnectionInformation (
   NWCONN_HANDLE   connHandle,
   nuint16         connNumber,
   pnstr8          pObjName,
   pnuint16        pObjType,
   pnuint32        pObjID,
   pnuint8         pLoginTime);
```

Delphi

```
uses calwin32

Function NWGetConnectionInformation
  (connHandle : NWCONN_HANDLE;
   connNumber : nuint16;
   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 object type of the client (optional).

pObjID

(OUT) Points to the 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 (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89FB	INVALID_PARAMETERS: Connection needs to be authenticated
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

Byte	Value	Range
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 (page 26), NWGetUserInfo (Server Management)

NWGetConnectionStatus (obsolete-moved from .h file 11/99)

Was last documented in September 1999. Call the NWCCGetConnRefInfo (page 42) and NWGetConnectionInformation (page 91) functions instead.

NWGetConnectionUsageStats (obsolete-moved from .h file 6/99)

was last documented in Release 15 for NetWare 2.x only.

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, 5.x, 6.x

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

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

Service: Connection

Syntax

C

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

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

Delphi

```
uses calwin32

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 (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x89FB	INVALID_PARAMETERS

NCP Calls

0x2222 23 31 Get Connection List From Object

NWGetDefaultConnectionID (obsolete-moved from .h file 11/99)

Was last documented in September 1999. Call the NWGetDefaultConnRef (page 99) and NWGetNearestDSConnRef (page 105) functions instead.

NWGetDefaultConnRef

Returns the default connection reference of the current session

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Connection

Syntax

C

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

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

Delphi

```
uses calwin32

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

Parameters

pConnReference

(OUT) Points to the default connection reference.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x880F	NO_CONNECTION_TO_SERVER

Remarks

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).

See Also

NWCCGetPrimConnRef (page 46), NWGetDriveStatusConnRef (Multiple and Inter-File Services)

3.4 NWGetl*-Z* Functions

Click on any function name in the table of contents to view the purpose, syntax, parameters, and return values for that function.

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: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Connection

Syntax

C

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

NWCCODE N_API  NWGetInetAddr (
   NWCONN_HANDLE        connHandle,
   nuint16              connNum,
   NWINET_ADDR  N_FAR  *pInetAddr);
```

Delphi

```
uses calwin32

Function NWGetInetAddr
  (connHandle : NWCONN_HANDLE;
   connNum : nuint16;
   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 (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_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 inetaddr.c (../../../samplecode/clib_sample/connect/inetaddr/inetaddr.c.html) for sample code.

NCP Calls

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

NWGetMaximumConnections (obsolete-moved from .h file 11/99)

Was last documented in September 1999. Call the NWCCGetNumConns (page 44) function instead.

NWGetNearestDirectoryService (obsolete-moved from .h file 11/99)

Was last documented in September 1999. Call the NWGetNearestDSConnRef (page 105) and NWCCOpenConnByRef (page 60) functions instead.

NWGetNearestDSConnRef

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

Local Servers: blocking

Remote Servers: blocking

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

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

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

Service: Connection

Syntax

C

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

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

Delphi

```
uses netwin32

Function NWGetNearestDSConnRef
 (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 (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8846	NWE_DS_NO_CONN

Remarks

If no NDS servers are found, NWE_DS_NO_CONN is returned.

See Also

NWCCGetAllConnRefInfo (page 28), NWCCGetConnRefInfo (page 42), NWCCOpenConnByRef (page 60), NWCCScanConnRefs (page 69), NWGetPreferredConnName (NDK: Novell eDirectory Core Services), NWSetPreferredDSTree (NDK: Novell eDirectory Core Services)

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: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Connection

Syntax

C

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

NWCCODE N_API NWGetObjectConnectionNumbers (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *pObjName,
   nuint16             objType,
   pnuint16            numConns,
   NWCONN_NUM N_FAR   *connList,
   nuint16             maxConns);
```

Delphi

```
uses calwin32

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

pObjName

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

objType

(IN) Specifies the 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 (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89F0	WILD_CARD_NOT_ALLOWED
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 above server, NWGetObjectConnectionNumbers will return SUCCESS and the `numConns` parameter will be zero indicating there are no connections with the server.

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)

NWGetTaskInformationByConn

Returns information about the active tasks assigned to the specified connection

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Connection

Syntax

C

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

NWCCODE N_API  NWGetTaskInformationByConn (
   NWCONN_HANDLE           conn,
   nuint16                 connNum,
   CONN_TASK_INFO  N_FAR  *taskInfo);
```

Delphi

```
uses calwin32

Function NWGetTaskInformationByConn
  (conn : NWCONN_HANDLE;
   connNum : nuint16;
   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 the CONN_TASK_INFO structure.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88FE	Unknown Packet Format
0x890A	NLM_INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89C6	NO_CONSOLE_PRIVILEGES
0x89FD	BAD_STATION_NUMBER

NCP Calls

0x2222 23 234 Get Connection's Task Information (3.x-6.x)

NWRequest

Passes an NCP request to the server

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Connection

Syntax

C

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

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

Delphi

```
uses calwin32

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

(IN) Points to the NCP function number.

numReqFrags

(IN) Points to the number of fragments pointed to by the `reqFrags` field (maximum is five).

reqFrags

(IN) Points to the list of request fragments.

numReplyFrags

(IN) Points to 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 (*Return Values for C*) 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.

SetConnectionCriticalErrorHandler

Specifies a function to handle connection timeout errors

Local Servers: blocking

Remote Servers: N/A

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

Connection Structures

4

This documentation alphabetically lists the Connection structures and describes their purpose, syntax, and fields.

CONN_TASK

Service: Connection

Defined In: nwmisc.h, calwin32, and nwncpext.txt

Structure

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

Delphi Structure

```
CONN_TASK = packed 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 calwin32

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;
```

Delphi Structure

```
CONN_TASK_INFO = packed 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;
```

Delphi Structure

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

Fields

`systemElapsedTime`

Specifies 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`

Specifies the number of bytes the associated connection has read.

`bytesWritten`

Specifies the number of bytes the associated connection has written.

`totalRequestPackets`

Specifies the number of requests the associated connection has made.

CONNECT_INFO

Returns connection status information

Service: Connection

Defined In: nwconnec.h, calwin32

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;
```

Delphi Structure

```
Defined in nwconnec.inc

  CONNECT_INFO = packed 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

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

connectFlags

Specifies the flag whose values are defined in the following table:

C Values	Delphi Values	Flag Name	Description
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 whether the connection is a NDS connection.
0x0400	$4000	CONNECTION_PNW	
0x8000	$8000	CONNECTION_AUTHENTICATED	Indicates if the connection is authenticated.

sessionID

Specifies 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

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

serverAddr

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

serverType

Specifies the server's bindery object type.

serverName

Specifies the server's bindery name.

clientType

Specifies the client' bindery object type.

clientName

Specifies the client's bindery object name.

NW_FRAGMENT

Fragments the request into the appropriate packet size

Service: Connection

Defined In: nwmisc.h and calwin32

Structure

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

Delphi Structure

```
  NW_FRAGMENT = packed 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 clxwin32

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 ;
#ifdef NWCC_INFO_VERSION_2
   pNWCCTranAddr   tranAddr ;
#endif
#ifdef NWCC_INFO_VERSION_3
   nuint32         identityHandle;
#endif
} NWCCConnInfo;
```

Delphi Structure

```
NWCCConnInfo = packed 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}
   {$IFDEF NWCC_INFO_VERSION_2}
      identityHandle : nuint32;
   {$ENDIF}
  End;
```

Fields

authenticationState

Specifies the NDS™ authenticated state of the specified connection (see Section 5.5, “NWCC_INFO_AUTHENT_STATE Values,” on page 137).

broadcastState

Specifies the message broadcast state of the specified connection (see Section 5.6, “NWCC_INFO_BCAST_STATE Values,” on page 137).

connRef

Specifies the connection reference for the specified connection handle.

treeName

Specifies the tree name of the specified connection if attached to NDS. It has a maximum length of NW_MAX_TREE_NAME_LEN.

connNum

Specifies the connection number for the specified connection.

userID

Specifies the user for the connection.

serverName

Specifies the name of the server the connection is attached to. It has a maximum length of NW_MAX_SERVER_NAME_LEN.

NDSState

Specifies if the connection supports NDS (see Section 5.8, “NWCC_INFO_NDS_STATE Values,” on page 138).

maxPacketSize

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

licenseState

Specifies if the connection is licensed (see Section 5.7, “NWCC_INFO_LICENSE_STATE Values,” on page 137).

distance

Specifies 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.

identityHandle

Specifies the identity of the connection.

Remarks

The `treeName` structure field is used to give the tree name of a particular connection in functions such as NWCCGetAllConnRefInfo (page 28) and NWCCGetConnRefInfo (page 42). 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.

NWCCFRAG

Fragments an NCP request into the appropriate packet size.

Service: Connection

Defined In: nwclxcon.h

Structure

```
typedef struct tagNWCCFrag
{
   nptr    address;
   nuint   length;
} NWCCFRAG, N_FAR *pNWCCFrag;
```

Fields

`address`

Points to the reply buffer.

`length`

Specifies the size of the buffer.

NWCCTranAddr

Defines the transport address for the specified connection

Service: Connection

Defined In: nwclxcon.h and clxwin32

Structure

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

Delphi Structure

```
Defined in nwclxcon.inc

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

Fields

`type`

(IN/OUT) Specifies the type of the transport address (see Section 5.12, “Transport Type Values,” on page 139).

`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

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 needed to return all the information.

Addresses using this structure are in printable order, and have a format that is the same as the format for the NWFSE_NETWORK_ADDRESS (Server Management) structure. The table below describes the address format:

Table 4-1 *Address Format*

TranType	Length	Address Format
IPX	12 bytes	Network (4 bytes) - The server's IPX Internal Network Number (ServerID) Node (6 bytes) - 0x00 0x00 0x00 0x00 0x00 0x01 Socket (2 bytes) - 0x04 0x51
TCP or UDP	6 bytes	For Windows NT/2000 • Socket (2 bytes) - 0x02 0x0C • IP Address (4 bytes), network (printable) order For Windows 95/98 and NLM • IP Address (4 bytes), network (printable) order • Socket (2 bytes) - 0x02 0x0C (omit on NLM, see note below)

NOTE: For the NLM platform, specifying the socket after an IP address is allowed but not necessary. It is recommended that you simply omit the socket value and specify a length of 4.

This information is clearer to understand with the following examples.

Suppose a server has an IPX Internal Network Number of 01012493. The NWCCTranAddr structure would be filled out as follows (an IP address would be filled out similarly, using the order from the table above):

```
NWCCTranAddr tranAddr;
nuint8 networkAddress[12];

tranAddr.type = NWCC_TRAN_TYPE_IPX;
tranAddr.len = 12;
tranAddr.buffer = networkAddress;

networkAddress[0] = 0x01;   /* Network Address */
networkAddress[1] = 0x01;
networkAddress[2] = 0x24;
networkAddress[3] = 0x93;
networkAddress[4] = 0x00;   /* Node */
networkAddress[5] = 0x00;
networkAddress[6] = 0x00;
networkAddress[7] = 0x00;
networkAddress[8] = 0x00;
networkAddress[9] = 0x01;
networkAddress[10] = 0x04;  /* Socket - Always 0x04, 0x51 for IPX */
networkAddress[11] = 0x51;
```

To connect to a server at the IP address 10.4.3.22, the NWCCTranAddr structure would be filled out as follows (Windows 95/98):

```
NWCCTranAddr tranAddr;
nuint8 networkAddress[6];
```

```
tranAddr.tranType = NWCC_TRAN_TYPE_TCP;
tranAddr.len = 6;
tranAddr.buffer = networkAddress;

/* Windows 95/98 Version */
networkAddress[0] = 10;   /* Network Address */
networkAddress[1] = 4;
networkAddress[2] = 3;
networkAddress[3] = 22;
networkAddress[4] = 0x02;  /* Socket - Always 0x02, 0x0C */
networkAddress[5] = 0x0C;
```

For NLM, the structure is filled out in the same way, but the length is shortened and the socket is not specified:

```
NWCCTranAddr tranAddr;
nuint8 networkAddress[4];

tranAddr.tranType = NWCC_TRAN_TYPE_TCP;
tranAddr.len = 4;
tranAddr.buffer = networkAddress;

/* NLM Version */
networkAddress[0] = 10;   /* Network Address */
networkAddress[1] = 4;
networkAddress[2] = 3;
networkAddress[3] = 22;
```

To connect to a server at the IP address 10.4.3.22, the NWCCTranAddr structure would be filled out as follows (Windows NT/2000):

```
NWCCTranAddr tranAddr;
nuint8 networkAddress[6];

tranAddr.tranType = NWCC_TRAN_TYPE_TCP;
tranAddr.len = 6;
tranAddr.buffer = networkAddress;

   /* Windows NT/2000 Version */
networkAddress[0] = 0x02;  /* Socket - Always 0x02, 0x0C */
networkAddress[1] = 0x0C;
networkAddress[2] = 10;    /* Network Address */
networkAddress[3] = 4;
networkAddress[4] = 3;
networkAddress[5] = 22;
```

NWCCVersion

Defines the NetWare server version of the connection

Service: Connection

Defined In: nwclxcon.h and clxwin32

Structure

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

Delphi Structure

```
Defined in nwclxcon.inc

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

Fields

major

Specifies the major version of NetWare. For example, `major` will be 4 for NetWare 4.1.

minor

Specifies the minor version of NetWare. For example, `minor` will be 12 for NetWare 3.12.

revision

Specifies an interim release number.

NWINET_ADDR

Returns the internet address for the specified connection

Service: Connection

Defined In: nwconnec.h and calwin32

Structure

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

Delphi Structure

```
 NWINET_ADDR = packed Record
    networkAddr : Array[0..3] Of nuint8;
    netNodeAddr : Array[0..5] Of nuint8;
    socket : nuint16;
    connType : nuint16;
  End;
```

Fields

networkAddr

Specifies the network address.

netNodeAddr

Specifies the network node address.

socket

Specifies the network socket in HI-LO byte order.

connType

Specifies the connection type used for NetWare 3.11 and above only (see Section 5.1, “Connection Type Values,” on page 135).

Remarks

On Intel-based platforms, the value in `socket` needs to be byte swapped from its original HI-LO order.

Connection Values

5

This documentation describes the values associated with Connection.

5.1 Connection Type Values

The following values defined in nwsfe.h indicate active connection types. Note that NCP has two connection types, one for IPX and one for IP.

Value	Connection Type	Comment
1	(Reserved for CLIB backward compatability)	
2	FSE_NCP_CONNECTION_TYPE	NCP connection over IPX
3	FSE_NLM_CONNECTION_TYPE	
4	FSE_AFP_CONNECTION_TYPE	
5	FSE_FTAM_CONNECTION_TYPE	
6	FSE_ANCP_CONNECTION_TYPE	
7	FSE_ACP_CONNECTION_TYPE	
8	FSE_SMB_CONNECTION_TYPE	
9	FSE_WINSOCK_CONNECTION_TYPE	
10	FSE_HTTP_CONNECTION_TYPE	
11	FSE_UDP_CONNECTION_TYPE	NCP connection over IP (NetWare 5.x or above)

5.2 Connection State Values

These values indicate the state of the connection and are set in the `openState` parameter:

C Value	Delphi Value	Value Name
0x0001	$0001	NWCC_OPEN_LICENSED
0x0002	$0002	NWCC_OPEN_UNLICENSED
0x0004	$0004	NWCC_OPEN_PRIVATE: For the Windows platform, the requester will ignore searching the connection table and always create a new connection. Such a connection can be seen only by the application that creates the connection. Not supported on NetWare. Using this value, it is possible to create more than one NCP connection on an Windows NT machine to the same server (while still being in the same Windows NT security context). However, you will be creating attached connections that cannot be used to access, for example, a file system.

C Value	Delphi Value	Value Name
0x0008	$0008	NWCC_OPEN_PUBLIC: For the Windows platform, the requester will first look for whether the connection to the server that is being requested is already established. If it is an established connection, the requester will return the handle to the existing connection; otherwise, the requester creates a new connection.
0x0010	$0010	NWCC_OPEN_EXISTING_HANDLE
0x0100	$0100	NWCC_OPEN_NEAREST
0x0200	$0200	NWCC_OPEN_IGNORE_CACHE

5.3 Feature Code Values

These are the feature code values:

C Value	Name
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

5.4 infoType Parameter Values

The NWCCGetConnInfo (page 35) and NWCCGetConnRefInfo (page 42) functions use one of the following flags in the `infoType` parameter to indicate the type of data to return:

C Value	Value	Minimum Buffer Size: Description
0x0000	NWCC_INFO_NONE	
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(49)
0x0008	NWCC_INFO_NDS_STATE	nuint

C Value	Value	Minimum Buffer Size: Description
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 (page 132)
0x000D	NWCC_INFO_TRAN_ADDR	sizeof NWCCTranAddr (page 129)
0x000E	NWCC_INFO_IDENTITY_HANDLE	nuint32
0xFFFF	NWCC_INFO_RETURN_ALL	Pointer to NWCCConnInfo (page 125)

5.5 NWCC_INFO_AUTHENT_STATE Values

NWCC_INFO_AUTHENT_STATE can return one of the following values:

C Value	Delphi 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

5.6 NWCC_INFO_BCAST_STATE Values

NWCC_INFO_BCAST_STATE can return one of the following:

C Value	Delphi 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

5.7 NWCC_INFO_LICENSE_STATE Values

NWCC_INFO_LICENSE_STATE can return one of the following:

C Value	Delphi Value	Value Name: Description
0x0000	$0000	NWCC_NOT_LICENSED: Connection is not licensed
0x0001	$0001	NWCC_CONNECTION_LICENSED: Connection is licensed

C Value	Delphi Value	Value Name: Description
0x0002	$0002	NWCC_HANDLE_LICENSED: Connection is scheduled to be licensed once it is authenticated

5.8 NWCC_INFO_NDS_STATE Values

NWCC_INFO_NDS_STATE can return one of the following:

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

5.9 Name Format Values

These are the name format values:

C Value	Delphi Value	Value Name
0x0001	$0001	NWCC_NAME_FORMAT_NDS: Used with NWCCOpenConnByName (page 55) and the Windows NT requester to specify an NDS object name as an open target
0x0002	$0002	NWCC_NAME_FORMAT_BIND: Used with a server name
0x0008	$0008	NWCC_NAME_FORMAT_NDS_TREE: Used with an NDS tree name
0x8000	$8000	NWCC_NAME_FORMAT_WILD: Used with either a tree or server name. Allows the system to determine the type of name that is passed to NWCCOpenConnByName.

5.10 Scan Flag Values

These are the scan flag values:

C Value	Value Name: Description
0x0000	NWCC_MATCH_NOT_EQUALS: Specifies every connection not matching a given data member should be scanned.
0x0001	NWCC_MATCH_EQUALS: Specifies every connection matching a given data member should be scanned.
0x0002	NWCC_RETURN_PUBLIC: Specifies all public connections should be considered in the scan.
0x0004	NWCC_RETURN_PRIVATE: Specifies all private connections should be considered in the scan.
0x0008	NWCC_RETURN_LICENSED: Specifies that only licensed connections are desired.
0x0010	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.

5.11 Security Flag Values

The following values may be ORed together:

Value	Name
0x00000000	NWCC_SECURITY_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.

5.12 Transport Type Values

These are the transport type values:

C Value	Delphi Value	Value Name
0x0001	$0001	NWCC_TRAN_TYPE_IPX
0x0003	$0003	NWCC_TRAN_TYPE_DDP
0x0004	$0004	NWCC_TRAN_TYPE_ASP
0x0008	$0008	NWCC_TRAN_TYPE_UDP
0x0009	$0009	NWCC_TRAN_TYPE_TCP

C Value	Delphi Value	Value Name
0x000A	$000A	NWCC_TRAN_TYPE_UDP6
0x000B	$000B	NWCC_TRAN_TYPE_TCP6
0x8000	$8000	NWCC_TRAN_TYPE_WILD

Connection Number and Task Management Concepts

6

This documentation describes Connection Number and Tasks Management, its functions, and features.

6.1 Overview

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 because a previously allocated connection number might be freed and available for another remote client.

6.2 Remote and Local Connections

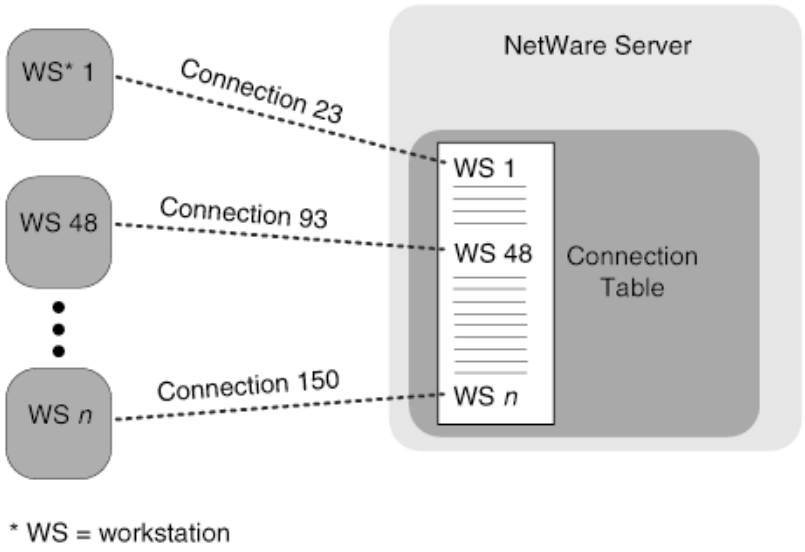
All connections are considered to be remote or local. A workstation connection is always a remote connection, since the machine it runs on is physically separate from the server. An NLM connection, on the other hand, can be remote or local. The connection is local when the NLM accesses the server the 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

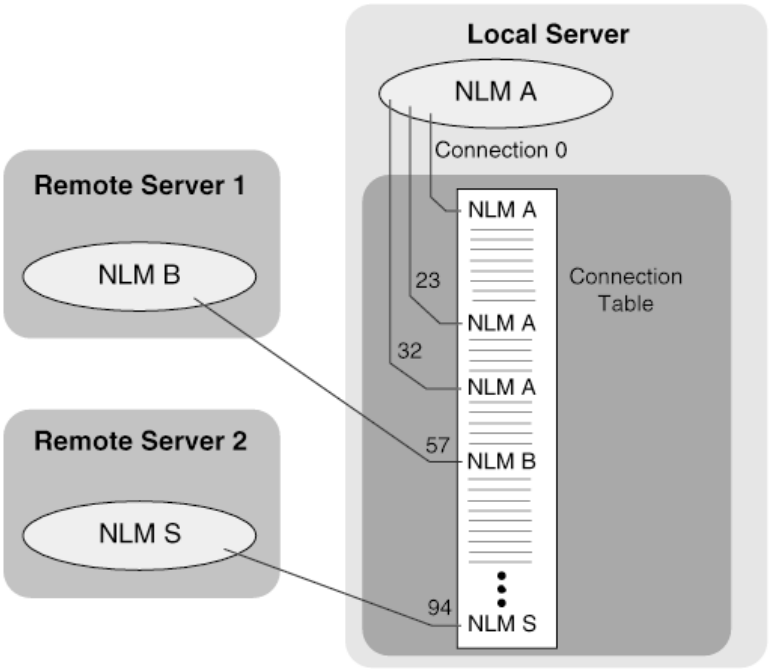
request to the server, which uses the number to verify security and carry out accounting and other functions.

Figure 6-1 *Remote Connections in the Connection Table*



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 6-2 *Local and Remote Connections in the Connection Table*



6.3 Task Numbers

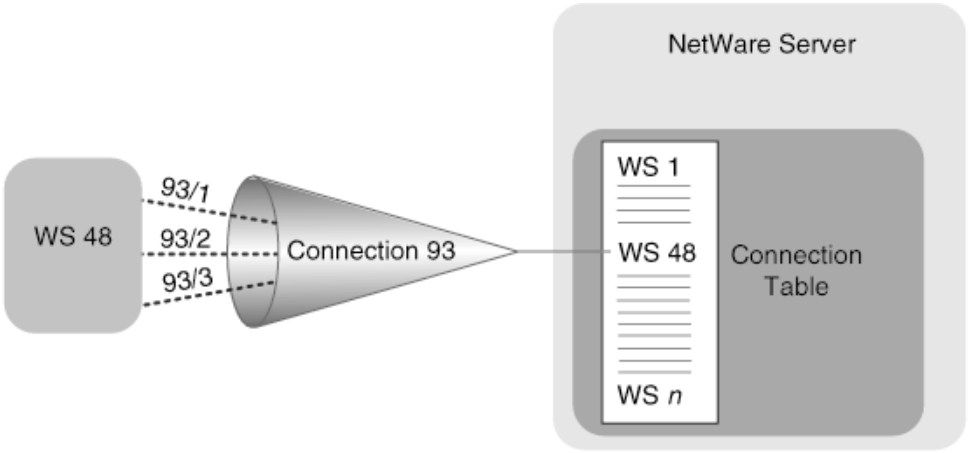
In NetWare, a task is a program running on a network workstation or server. NetWare uses a task number to identify each task.

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 6-3 *Running Multiple Sessions on a Connection*



6.4 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 (Server Management) or NWAttachToFileServer (Server Management) (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.

6.5 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 (page 166) 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 (page 169) 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.

6.6 Connection Numbers

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)

6.7 Connection Zero

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

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 insecure 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.

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.

6.8 Proxy Work

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.

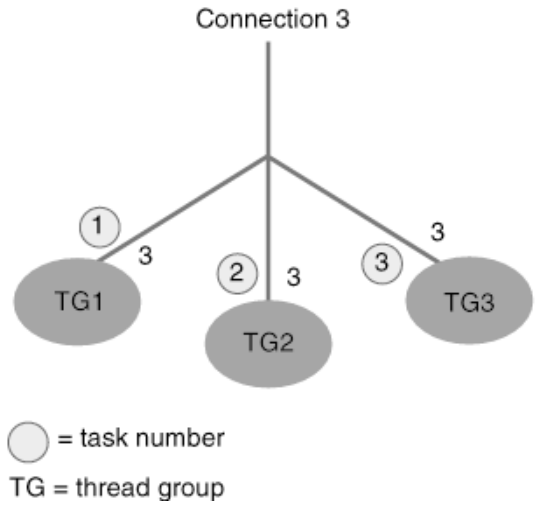
An NLM can do work by proxy in a couple of ways. It can either get a connection and log an object of its choosing in on that connection, or it can temporarily take over the client's existing connection to the local server.

6.9 Multiple Thread Groups on a Single Connection

You could give multiple thread groups the same connection number with SetCurrentConnection (page 166). Then, with SetCurrentTask (page 169), you could assign each of them a different task number 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.

Figure 6-4 *Thread Groups and Task Numbers*



To reduce the danger of duplicating a connection/task pair, call AllocateBlockOfTasks (page 152) 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.

6.10 Connection Number and Task Management Functions

These function descriptions use the terms station, connection, and connection number interchangeably:

Function	Description
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.

Connection Number and Task Management Tasks

7

This documentation describes common tasks associated with Connection Number and Task Management.

7.1 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 (page 160).

DisableConnection (page 154) 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.

7.2 Intervening on an Established Connection

Call DisableConnection (page 154) 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.

7.3 Doing Work on a Single Connection

One advantage of using a single connection is that it 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 test 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 (Server Management).

7.4 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 (page 154) to temporarily reserve the workstation's connection number solely for the use of the NLM. Call EnableConnection (page 156) to reverse the effect of DisableConnection (page 154). 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 (page 169).

7.5 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 (page 160), 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 NWLoginToFileServer (Server Management).

7.6 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 (page 169). It sets the current task to the value you pass it as `taskNumber`.

You can also use SetCurrentTask (page 169) 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 (page 152). 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 (page 164) to free the task numbers before unloading.

7.7 Servicing a Single Connection With 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).

Connection Number and Task Management Functions

8

This documentation alphabetically lists the Connection Number and Task Management functions and describes their purpose, syntax, parameters, and return values.

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

AllocateBlockOfTasks

Returns a unique set of task numbers for the exclusive use of the requesting NLM

Local Servers: nonblocking

Remote Servers: nonblocking

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

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 above 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 (page 164), SetCurrentTask (page 169)

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, 5.x, 6.x

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.

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, 5.x, 6.x

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

The following table lists return values and descriptions.

Value	Hex	Name	Description
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 (page 153),

EnableConnection

Enables the server to service requests for the specified connection

Local Servers: nonblocking

Remote Servers: N/A

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

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

The following table lists return values and descriptions.

Value	Hex	Name	Description
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 (page 154)

GetCurrentConnection

Returns the current connection on the current server

Local Servers: nonblocking

Remote Servers: nonblocking

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

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. For information about connection numbers, see Section 6.2, “Remote and Local Connections,” on page 141.

See Also

SetCurrentConnection (page 166)

GetCurrentFileServerID

Returns the current file server ID number

Local Servers: nonblocking

Remote Servers: nonblocking

Classification: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Service: Connection Number and Task Management

Syntax

```
#include <nwcntask.h>

WORD GetCurrentFileServerID  (void);
```

Return Values

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.

See Also

SetCurrentFileServerID (page 168)

GetCurrentTask

Returns the calling thread group's current task number

Local Servers: nonblocking

Remote Servers: nonblocking

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

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 (page 169)

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, 5.x, 6.x

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) Points to the string containing the name of the object.

objectType

(IN) Specifies the type of the object.

password

(IN) Points to the string containing the object's password (lowercase passwords can be specified).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name and description
0	0x00	ESUCCESS: Logout of object was successful
-1		EFAILURE: Connection number not valid
150	0x96	ERR_SERVER_OUT_OF_MEMORY

Remarks

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:

Table 8-1 *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®.

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.

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 (Server Management), LogoutObject (page 163)

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, 5.x, 6.x

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

The following table lists return values and descriptions.

Value	Hex	Name	Description
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.

See Also

LoginObject (page 160)

ReturnBlockOfTasks

Frees a block of task numbers

Local Servers: blocking

Remote Servers: nonblocking

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

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

The following table lists return values and descriptions.

Value	Hex	Name	Description
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 (page 152), SetCurrentTask (page 169)

ReturnConnection

Returns a connection number the NLM previously allocated

Local Servers: blocking

Remote Servers: blocking

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

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

The following table lists return values and descriptions.

Value	Hex	Name	Description
0	(0x00)	ESUCCESS	Connection number returned.

Remarks

ReturnConnection returns a connection number previously allocated by SetCurrentConnection, LoginObject, NWAttachToFileServer, or LoginToFileServer.

See Also

NWAttachToFileServer (Server Management), LoginObject (page 160), NWLoginToFileServer (Server Management), SetCurrentConnection (page 166)

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, 5.x, 6.x

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 (page 157), ReturnConnection (page 165)

Example

```
#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();
}
```

SetCurrentFileServerID

Sets the current connection ID (server ID)

Local Servers: N/A

Remote Servers: blocking

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

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 (page 158)

SetCurrentTask

Sets the calling thread group's current task number

Local Servers: nonblocking

Remote Servers: nonblocking

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

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 (page 152), GetCurrentTask (page 159), ReturnBlockOfTasks (page 164)

Message Concepts

This documentation describes Message, its functions, and features.

9.1 Message Modes

A workstation has a configurable message mode on the NetWare server to which it's connected. The message mode enables and disables the reception of messages, and lets the workstation discriminate between messages from other users and messages from the server console.

The mode also lets you control the notification feature that causes the workstation software to retrieve a message automatically. If notification is disabled, your application must poll the server for current messages. The following table shows possible values for the message mode. The default value is 0, enabling all broadcasts.

Table 9-1 *Message Modes*

Mode	Value	Comment
Enable all	0	Receive all broadcasts.
Server only	1	Receive only server broadcasts. Discard user broadcasts.
Disable all	2	Disable all broadcasts. Discard user broadcasts. Store server broadcast but don't notify.
Disable notify	3	Store both user and server broadcasts but don't notify.

9.2 Message Size

All broadcast messages should be NULL-terminated. The message size and the number of connections to which you can send messages depends on the version of the server.

- For NetWare 3.11b and below, a message can be from 1 to 58 bytes long including the null terminator and can be sent to between 1 and the maximum number of possible connections (configurable up to 256).
- For NetWare 3.11c and above, the message can be 1 to 254 bytes long including a null terminator and be sent to as many as 62 connections.

When retrieving a message on networks running NetWare 3.11c and above, allocate a buffer at least 254 bytes in length.

You can also send messages to the NetWare server console. The message is displayed in a single line on the console screen after the colon (:) prompt. The NetWare SEND, CASTON, and CASTOFF utilities are examples of how to apply these functions. (CASTON and CASTOFF are NetWare 3.11 utilities.)

9.3 Message Functions

These functions send and receive broadcast messages. They are declared in nwmsg.h. It is possible only a subset of these functions are supported by a specific client.

Function	Description
NWBroadcastToConsole	Sends a message to the default NetWare server's system console.
NWDisableBroadcasts	Informs the server that a client doesn't want to receive messages from other clients.
NWEnableBroadcasts	Allows a client to receive broadcast messages after broadcasts have been disabled.
NWGetBroadcastMessage	Returns a message from the specified NetWare server. (Not supported on Unix.)
NWSendBroadcastMessage	Sends a message to the specified logical connections on the specified NetWare server.
NWSendConsoleBroadcast	Sends a console message to the specified logical connection. The functions requires operator rights.
NWSetBroadcastMode	Sets the message mode for the workstation on the specified NetWare server.

Message Functions

10

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

NWBroadcastToConsole

Sends a message to the default server's system console

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Message

Syntax

C

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

NWCCODE N_API NWBroadcastToConsole (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *message);
```

Delphi

```
uses calwin32

Function NWBroadcastToConsole
  (conn : NWCONN_HANDLE;
   const message : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

message

(IN) Points to the NULL-terminated message to be sent.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

0x890A	NLM_INVALID_CONNECTION

Remarks

Under NETX, if an invalid connection handle is passed to `conn`, NWBroadcastToConsole will return 0x0000. NETX will pick a default connection handle if the connection handle cannot be resolved.

The message is displayed in a single line on the console screen after the colon (:) prompt. Messages longer than 58 bytes are truncated without notifying the broadcasting workstation. New messages overwrite previous messages at the console.

NCP Calls

0x2222 21 9 Broadcast To Console

NWDisableBroadcasts

Informs the server a client does not want to receive messages from other client

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Message

Syntax

C

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

NWCCODE N_API  NWDisableBroadcasts  (
   NWCONN_HANDLE   conn);
```

Delphi

```
uses calwin32

Function NWDisableBroadcasts
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x89FF	Broadcast Disabled

Remarks

After calling NWDisableBroadcasts, the server does not allow other clients to log messages for forwarding to this client. If another client attempts to broadcast to a client with broadcast disabled, 0x89FF (failed) is returned. NWDisableBroadcasts can be used by any client.

NCP Calls

0x2222 21 2 Disable Broadcasts

See Also

NWEnableBroadcasts (page 178)

NWEnableBroadcasts

Allows a client to enable message reception after broadcast reception has been disabled by calling NWDisableBroadcasts

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Message

Syntax

C

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

NWCCODE N_API  NWEnableBroadcasts  (
   NWCONN_HANDLE   conn);
```

Delphi

```
uses calwin32

Function NWEnableBroadcasts
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL

Remarks

Messages are enabled by default when the connection is first established. NWEnableBroadcasts can be called by any client.

NCP Calls

0x2222 21 3 Enable Broadcasts

See Also

NWDisableBroadcasts (page 176)

NWGetBroadcastMessage

Returns a message from the specified server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Message

Syntax

C

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

NWCCODE N_API  NWGetBroadcastMessage  (
   NWCONN_HANDLE   conn,
   pnstr8          message);
```

Delphi

```
uses calwin32

Function NWGetBroadcastMessage
  (conn : NWCONN_HANDLE;
   message : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

message

(OUT) Points to the message buffer where the message will be stored.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x89FD	BAD_STATION_NUMBER

Remarks

Because some servers support 256-byte messages, the message buffer passed in should be large enough to contain messages of this size.

NCP Calls

0x2222 21 01 Get Broadcast Message
0x2222 21 11 Get Broadcast Message (new)
0x2222 23 17 Get File Server Information

NWSendBroadcastMessage

Allows a client to send a broadcast message to the specified logical connections on the specified NetWare server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Message

Syntax

C

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

NWCCODE N_API NWSendBroadcastMessage (
   NWCONN_HANDLE          conn,
   const nstr8 N_FAR     *message,
   nuint16                connCount,
   const nuint16 N_FAR   *connList,
   pnuint8                resultList);
```

Delphi

```
uses calwin32

Function NWSendBroadcastMessage
  (conn : NWCONN_HANDLE;
   message : pnstr8;
   connCount : nuint16;
   connList : pnuint16;
   resultList : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

message

(IN) Points to the NULL-terminated message being sent.

connCount

(IN) Specifies the number of connections in the connection list.

connList

(IN) Points to an array containing the connection numbers of all stations scheduled to receive the message.

resultList

(OUT) Points to an array containing result codes for all stations being sent the broadcast.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x89FB	INVALID_PARAMETERS
0x89FC	MESSAGE_QUEUE_FULL
0x89FD	BAD_STATION_NUMBER

Remarks

NWSendBroadcastMessage can be used by any client. The specified NetWare server attempts to store the broadcast message in the message buffer of each target connection. A result code for each target is returned by NWSendBroadcastMessage in `resultList`. Valid result codes are listed below:

Result code	Description
0x0000	Successful. NetWare server stored the message in the target connection's message buffer.
0x0001	(4.0 only) Illegal station number-station number (conn) is invalid
0x0002	(4.0 only) Client not logged in-intended recipient of the message is not currently logged in to the default server, even though the client may be logged in to the network.
0x0003	(4.0 only) Client not accepting message-intended recipient of message not accepting incoming messages
0x0004	(4.0 only) Client already has message-server already has a message stored for intended recipient and cannot accept another message until that recipient clears the message from their screen
0x0096	(4.0 only) No allocation of space for the message on the server-message cannot be sent

These result codes indicate whether the NetWare server has successfully placed the message in the message buffer of the target connection. The NetWare server notifies the connection when a message arrives. However, placing the message in the message buffer and notifying the connection

does not guarantee that the target station received the message. It is the target's responsibility to retrieve and display the message, depending on the broadcast mode of the connection.

A broadcast message can have the following sizes:

```
before 3.11         1-58 bytes
3.11 and later      1-250 bytes
```

A broadcast can be sent to the following maximum number of configured connections:

```
before 3.11         1-200
3.11 and later      1-62
```

Messages longer than the appropriate buffer size are truncated. The broadcasting workstation does not receive a message regarding truncated broadcasts.

NCP Calls

0x2222 21 00 Send Personal Message (3.11a or below)
0x2222 21 10 Send Personal Message (3.11b or above)
0x2222 23 17 Get File Server Information

NWSendConsoleBroadcast

Broadcasts a message to the specified logical connections on the specified NetWare server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Message

Syntax

C

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

NWCCODE N_API NWSendConsoleBroadcast (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *message,
   nuint16             connCount,
   pnuint16            connList);
```

Delphi

```
uses calwin32

Function NWSendConsoleBroadcast
  (conn : NWCONN_HANDLE;
   message : pnstr8;
   connCount : nuint16;
   connList : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

message

(IN) Points to the NULL-terminated message being broadcast.

connCount

(IN) Specifies the number of connections in the connection list.

connList

(IN) Points to an array containing the connection number of all stations scheduled to receive the message.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x890A	NLM_INVALID_CONNECTION
0x89C6	NO_CONSOLE_PRIVILEGES
0x89FD	BAD_STATION_NUMBER

Remarks

The requesting client must have operator rights to call NWSendConsoleBroadcast.

Messages are not received by workstations that have disabled broadcasts or workstations that are not logged in. If `connCount` is set to 0, the message is sent to all connections.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 209 Send Console Broadcast
0x2222 23 253 Send Console Broadcast

See Also

NWSetBroadcastMode (page 187)

NWSetBroadcastMode

Sets the message mode of the requesting workstation

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: Message

Syntax

C

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

NWCCODE N_API  NWSetBroadcastMode  (
   NWCONN_HANDLE   conn,
   nuint16         mode);
```

Delphi

```
uses calwin32

Function NWSetBroadcastMode
  (conn : NWCONN_HANDLE;
   mode : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

mode

(IN) Specifies the broadcast mode to be set.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL
0x8836	INVALID_PARAMETER

Remarks

NWSetBroadcastMode can be used by any client.

When a broadcast message is sent, the NetWare server attempts to store the message in the message buffer of each target connection. The result of this action depends on the broadcast mode of the target station. The default mode is 0x00, receive all broadcasts. However, broadcast modes can be set to any one of the following by calling NWSetBroadcastMode :

Broadcast Mode	Description
0x0000	Receive all broadcasts (default mode).
0x0001	Receive only server broadcasts. User messages are not stored.
0x0002	Disable all broadcasts. User messages are not stored. Server messages are stored but notification is not given to the workstation.
0x0003	Both user and server messages are stored but message notification is not sent to the workstation. The client can poll for messages.

NOTE: When using NETX, NWCONN_HANDLE is ignored and NWBROADCAST_MODE is set for all logged-in connections.

NCP Calls

0x2222 21 02 Disable Broadcasts
0x2222 21 03 Enable Broadcasts

See Also

NWSendConsoleBroadcast (page 185)

NCP Extension Concepts

11

This documentation describes NCP Extension, its functions, and features.

With NCP Extension, the client can be either a workstation application or an NLM acting as a client. The server is an NLM running on a NetWare server.

The fundamental NetWare® services are provided by a set of functions implemented by the NetWare Core Protocol (NCP) software. Each routine is referred to as an NCP. Many of the NLM C Interface functions call the NCP routines.

NetWare allows you to register the services of an NLM as an NCP Extension, allowing you to extend the services provided by the NetWare OS while maintaining the advantages associated with NCPs. The main advantages of NCP Extension follow:

- Easy to use.
- Use an existing connection with a server (eliminating the need to establish a separate communications session with the server).
- Allow use of arbitrary message sizes

There are two sides to NLM Extensions:

- The service-providing side runs as an NLM on a server and registers its service as an NCP Extension. The NLM must be loaded on each server that provides the NCP Extension. An NLM that is loaded on one server cannot register an NCP Extension on a remote server.
- The client side uses the service of the NLM by calling the NCP Extension. The client can be an NLM acting as a client, or it can be an application running on a workstation.

NCP Extension is a client-server paradigm, where the following events occur:

- The client sends an NCP request to the server.
- The NCP Extension on the server processes the request.
- The server sends the results back to the client.

11.1 Client-Server Applications

NCP Extension is an excellent use for client-server applications, which allow the service-providing NLMW-close to the resource-to do the work for the client. For example, with a database, the client could send a request to the NCP Extension to search the database for a certain record. The function registered as the NCP Extension handler would interpret the request, process the search, and return the related information to the client

11.2 IPX/SPX Alternative

While NCP Extension does not replace every need for IPX and SPX, there are some cases where NCP Extension can simplify the communication between the client and the server. The advantage of an NCP Extension is it uses the existing connection of the client, freeing you from needing to set up communication sockets. You can use NCP Extension in many of the cases where you are currently using IPX or SPX.

The disadvantage is that NCP Extension takes a connection. If your application isn't already establishing a NetWare connection, and you don't want to establish one, you may choose to use IPX and SPX instead.

Another disadvantage to NCP Extension is that communication must always be initiated by the client. With IPX and SPX the client and/or the server can initiate communication.

11.3 Extension Context

The `NCPExtensionHandler`, `ConnectionEventHandler`, and `ReplyBufferManager` parameters of NWRegisterNCPExtension are registered as callbacks. These callbacks run as OS threads and are not able to call most of the NetWare API functions, unless they are given CLIB context. If context is not given to these callbacks and they call functions that need context, the server abends.

With the NetWare 4.x and above OS, threads have been given a context specifier that determines what CLIB context is given to the callbacks they register. You can determine the existing setting of the registering thread's context specifier by calling GetThreadContextSpecifier. Call SetThreadContextSpecifier to set the registering thread's context specifier to one of the following options:

- NO_CONTEXT-Callbacks registered by threads with this option set are not given CLIB context. The advantage here is that you avoid the overhead needed for setting up CLIB context. The disadvantage is that without the context, the callback is not able to use NetWare API functions that require thread group level context.
- USE_CURRENT_CONTEXT-Callbacks registered with a thread that has its context specifier set to USE_CURRENT_CONTEXT have the thread group context of the registering thread. This is the default setting for threads that are started with BeginThread, BeginThreadGroup, or ScheduleWorkToDo.
- A valid thread group ID-This is to be used when you want the callbacks to have a different thread group context than the thread that schedules them.

Although CLIB context can be given to these callbacks automatically (with the NetWare 4.x and above OS) by setting the registering thread's context specifier to USE_CURRENT_CONTEXT, your NCP Extension processes faster if you set the context specifier to NO_CONTEXT and then manually establish the context inside your callback by calling SetThreadGroupID, and passing in the ID of a valid thread group. (Note: This behavior is peculiar to the NCP Extension-handling code, and does not apply to callbacks in general.)

NLM applications that run on the NetWare 3.11 OS must manually set the thread-group syntax within the callbacks, by calling SetThreadGroupID and passing in a valid thread group ID.

For more information on using CLIB context, see Context Problems with OS Threads (*NDK: NLM Threads Management*).

11.4 Extension ID

After an NCP Extension is registered with the OS, it is available to service requests from clients. Before using an NCP Extension, the clients must verify that an NCP Extension is active for the service they want to use. An NCP Extension is identified by name or by ID.

An ID is also used to identify an NCP Extension. The following rules apply to an NCP Extension ID.

- They are unique.
- They are dynamically assigned by the OS when an NLM registers an NCP Extension with NWRegisterNCPExtension. These dynamic ID's are determined by the OS on a first-come, first-served basis, and they can be different each time an NCP Extension is registered.

  If an NCP Extension that is using dynamic IDs is deregistered and then registered again, it has a different ID. NCP Extension IDs increase in a monotonic manner. For example, if IDs 1 through 5 are used and the NCP Extension with an ID of 3 is deregistered and then reregistered, it will have an ID greater than 5. The ID 3 is not used again until the server is brought down and restarted.
- They can be "well known" IDs that NLM applications use to identify an NCP Extension when they register the Extensions with NWRegisterNCPExtensionByID. These IDs are the same each time an NCP Extension is registered, so they can be used to identify a specific NCP Extension.

NOTE: Well known IDs are assigned by Developer Support to guarantee uniqueness and that the IDs are in a valid range. Dynamically assigned IDs are not assigned by Developer Support since these IDs are not attached to a specific NCP Extension.

11.5 Extension Name

Every NCP Extension must have an identifying name. The following rules apply for naming an NCP Extension:

- Case-sensitive.
- Any text character string up to 32 bytes long, not counting the NULL terminator.
- Unique. It should be cleared through Developer Support to guarantee uniqueness.

NOTE: Problems can occur if two service providing NLM applications use the same name for each NCP Extension. The clients accessing the extensions by calling NWGetNCPExtensionInfo and NWScanNCPExtensions would not know if the extension they see registered is the one they want. To avoid duplicate names, you should clear your NCP Extension name through Developer Support.

11.6 Extension Security

IMPORTANT: You must make sure that your NCP Extension does not violate the security of the network.

Your service-providing NLM may have supervisor access to the server it is running on. If your NCP Extension handler provides a service that is sensitive to NetWare security issues (accesses requires NetWare security controls), it should take over the client's connection and make its requests using the client's connection. This helps ensure that the request is processed with the client's rights.

The following code fragment shows how to take over a client's connection:

```
// The current connection ID is assumed to be set to the local server.
oldConnection = SetCurrentConnection(clientsConnection);
```

```
// Server requests are made on behalf of the client, using his
// connection's security restrictions.
setCurrentConnection(oldConnection);
```

11.7 Extension Views

There are two views for NCP Extension: the client view and the server view.

11.7.1 Client View

The view from the client is different than that from the NCP Extension. The client sends requests and receives replies. It does not need to know the details of how the NCP Extension works; it only needs to know the protocol for the communication between them.

The client accesses the services of an NCP Extension in the following ways:

- Checks to see if the NCP Extension has been registered. A client cannot use an NCP Extension until it has been registered. The client can use NWGetNCPExtensionInfo or NWScanNCPExtensions to obtain the NCP Extension IDs of extensions that have been registered. If the NCP Extension ID is a well known ID it is not necessary to scan or get extension information because attempting to use the ID will return a failure if the extension is not registered.
- Sends a request to the NCP Extension with NWSendNCPExtensionRequest or NWSendNCPExtensionFraggedRequest and use the information that was returned.
- Asks for the information in an NCP Extension's query data buffer by calling NWGetNCPExtensionInfo or NWScanNCPExtensions and uses the query data that is returned.

11.7.2 Provider View

The NCP Extension does not need to know what the client process looks like; it only needs to know the format of the request coming from the client and how to format the reply.

The NCP Extension does the following:

- Registers the NCP Extension with the NCP Extension handler, the reply buffer manager, the connection event handler, and query data buffer.
- When the NCP Extension handler is called, it finds the request in a buffer that the OS allocated when the request was received. The NCP Extension handler processes the request and places the reply in another buffer(s) that the OS will use when sending the reply to the requester.
- If a reply buffer manager is used, it is called after the data in the reply buffer(s) has been sent to the client. When the OS calls the reply buffer manager the reply buffer address is passed to it and the reply buffer manager determines what to do with the buffer(s) where the reply is stored.
- When the connection event handler is called, it determines if the event affects the NCP Extension, and takes appropriate action.
- Updates the information in the query data buffer if there is a need..
- Deregisters the NCP Extension handler when it no longer wants to provide the service or when the service-providing NLM is unloaded.

11.8 Server Components

The NCP Extension server resides on a NetWare server and consists of the following components:

- NCP Extension handler (optional)
- Reply buffer manager (optional)
- Connection event handler (optional)
- Query data buffer

The NCP Extension handler is a routine that runs on the server and is called by the NetWare API whenever the client calls NWSendNCPExtensionRequest, or NWSendNCPExtensionFraggedRequest. The NCP Extension handler interprets the message sent by the client, processes the request, and sends a reply to the client.

A reply buffer manager is useful when the data to be transferred is already gathered or if the data should be kept in a specific memory location. The reply buffer manager is a routine that determines what to do with the reply buffer after the information in the buffer has been sent to the client. The OS calls this routine after it has sent the information in the buffer. Whether to use a reply buffer manager or not is a question of performance and function. The reply buffer manager might do things such as free the reply buffer, return it to a free list of buffers or unlock the data; the implementation is determined by the NCP Extension handler and the reply buffer manager.

The connection event handler is a routine that the server calls when any connection on the server is freed, killed, logged out, or restarted. One of the parameters to this routine is the connection that the event is happening on, and the other parameter is the event type. The connection event handler can use this information to determine if the connection belongs to a client that is being serviced by the NCP Extension handler, and if so, what action to take to clean up that connection's state.

The query data buffer is a 32-byte buffer that can be used to return information when NWGetNCPExtensionInfo or NWScanNCPExtensions is called. Calling these functions returns the contents of the update buffer to the client, which provides a one-way, passive information passing scheme.

NOTE: The query data buffer becomes the sole communication mechanism if an NCP Extension handler is not registered.

11.9 Data Transfer

The data that is used by the client and the server is stored in buffers as it moves through the process. For example, the client can store its information in one location or in up to four locations. The data stored in multiple locations is known as fragmented data.

If the client wants to send data that is stored in one buffer, it sends the request by calling NWSendNCPExtensionRequest. If the client wants to send fragmented data, it sends the request by calling NWSendNCPExtensionFraggedRequest.

NWSendNCPExtensionFraggedRequest gathers the data from the multiple locations and sends it as a stream of bytes to the server, just as if the data had come from one location.

Once the server has received all the data, the NetWare API calls the NCP Extension handler, giving the handler the address of the request buffer where the client's request is stored. The client's request is stored in one buffer, even if the client's request has come from multiple locations on the client.

After the NCP Extension handler has serviced the request, it can return the reply from one buffer or from multiple buffers (fragmented data as with the client). In either case, the reply is sent as a stream of bytes to the client.

When the reply returns to the client, the client's code is still within the NWSendNCPExtensionRequest or within the NWSendNCPExtensionFraggedRequest function. These functions place the data in the buffer(s) specified as parameters to the functions. NWSendNCPExtensionRequest places the reply in one buffer. NWSendNCPExtensionFragged Request can place the reply in one buffer, or it can separate the reply, placing it in multiple buffers.

11.10 Reentrancy

You must make sure your NCP Extension handler is reentrant. You cannot be assured that your NCP Extension handler runs to completion before it is called again by another client. Because more than one request can be accessing the same code, you need to code with reentrancy in mind. Similar issues are of concern exist for the reply buffer manager and the connection event handler as well. For more information about reentrancy, see Shared Memory (*NDK: NLM Threads Management*).

11.11 Reply Buffer Manager

The reply buffer manager is a routine that is called after the NCP Extension handler's reply to the client is sent. If you are going to use a reply buffer manager, you specify it when you register the NCP Extension with the OS.

If you specify that your NCP Extension uses a reply buffer manager, the NetWare API does not allocate reply buffers for your NCP Extension. In this case, the creation of the buffers is the responsibility of the NCP Extension handler.

The reply buffer manager does not allocate reply buffers. However, it can free the buffers that the NCP Extension handler allocates or manipulate data which controls access to those buffers.

The main reason for using a reply buffer manager is to avoid needless copying of reply data, thereby speeding up your application. It also minimizes possible failures due to Alloc Memory failures for copying data into another buffer when the data already exists in memory.

For example, if your NCP Extension handler maintains a buffer itself, it can save a copy cycle by returning a pointer to its buffer, rather than copying the buffer's contents into a buffer created by the NetWare API. If your NCP Extension is a game that maintains a screen buffer and returns the updated screen to the client after its player is moved, it would be best to send the screen data directly from the buffer it is maintained in.

Another example is with an NCP Extension that returns fragmented data. In this case the NCP Extension could have a routine that is constantly polling the server and placing information into various buffers. When the NCP Extension is called, the NCP Extension handler simply returns a structure that has fields pointing to the buffers where the information is located. This avoids copying the data from various locations and placing it in a single buffer.

Another reason for using a reply buffer manager is that, in some ways, it can be thought of as the second part of the NCP Extension handler. With the example in the previous paragraph, the NCP Extension handler could set a semaphore to stop the update routine from updating the buffers. Then, after the information in the buffers has been sent to the client, and the reply buffer manager is called, the reply buffer manager can reset the semaphore, allowing the update routine to continue with updating the buffers.

A common issue when using the reply buffer management scheme presented above is that of associating the call to the reply buffer manager with the associated call to the extension handler. The parameters to each callback are helpful in this regard. Even though the reply buffer address passed between calls is the same, this is sometimes insufficient. The connection number and task number are the same between calls, but this knowledge alone may require an additional private tracking mechanism to correctly associate the two callbacks.

To help eliminate the problem the NCPExtensionClient parameter has been constructed so that the same address is passed to both callbacks and the same contents are passed to both callbacks. In conjunction with this your extension handler can overwrite the two LONG members of the NCPExtension client structure with any values you like. These same values will then be returned to your reply buffer manager handler in the NCPExtension client parameter. This should be sufficient to allow you to accurately and efficiently coordinate the reply buffer between the extension handler and the reply buffer manager callbacks.

One other tip is that the reply buffer manager will not be called if the extension handler returns a nonzero return code. It will also not be called if no data was returned and REPLY_BUFFER_IS_FRAGGED was not used.

11.12 Connection Status

Your `connectionEventHandler` can keep track of when connections are freed, killed, logged out, or restarted. If keeping track of connection status is not important to you, you can pass NULL for the `ConnectionEventHandler` when you register the NCP Extension.

In some cases, this information is important; in other cases, it is not. A service that has a limit on the number of users would be interested in knowing when a connection was terminated, so it could allow another user to have access to the service. A service that allows unlimited access may not be concerned with who is using it.

IMPORTANT: If you are using a reply buffer manager, you should use a connection event handler. This is because the reply buffer manager is never called if the current client's connection goes down while the reply buffer is being sent to the client. Instead of calling the reply buffer manager, the OS eventually calls the connection event handler for that connection. It is then the responsibility of the connection event handler to recognize that the client has gone away and to free its resources accordingly.

Your connection event handler is called when a connection is freed, killed, logged out, or restarted. For the NetWare 3.12 OS and above, this information in received in the `eventType` parameter. This parameter may be tested for the following values:

- CONNECTION_BEING_FREED-This is returned when the client calls a function to return its connection, or an NLM is unloaded without returning its connection, or an attempt to create a connection fails.
- CONNECTION_BEING_KILLED-This means that someone has asked to kill the connection either explicitly or via a call to bring down the server.
- CONNECTION_BEING_LOGGED_OUT-The client has made a call to log out.
- CONNECTION_BEING_RESTARTED-This is returned when the client is making a call to create a connection when it has not already freed the connection. This can happen when the client station locks up and is rebooted. When the client tries to log in to the server, the server sees that the client is trying to allocate a connection when it already has a connection. The

server issues a notice of CONNECTION_BEING_RESTARTED, then a notice for CONNECTION_BEING_LOGGED_OUT. If the logout fails, the server issues a CONNECTION_BEING_FREED notice.

NOTE: The eventType parameter is not used for the NetWare 3.11 OS and previous versions. The prototype for the ConnectionEventHandler does not include this parameter. Do not attempt to interpret the eventType value if running on those versions of the OS.

11.13 NCP Extension Functions

Table 11-1 *NLM Service Provider Functions*

Function	Description
NWDeRegisterNCPExtension	Remove an NCP Extension from the OS.
NWRegisterNCPExtension	Using a specific name, register an NCP Extension with the OS
NWRegisterNCPExtensionByID	Using a specific ID and name, register an NCP Extension with the OS.

Table 11-2 *NLM Client Functions*

Function	Description
NWGetNCPExtensionInfo	Return information about the NCP Extension associated with a specific name.
NWGetNCPExtensionInfoByID	Return information about the NCP Extension associated with a specific ID.
NWScanNCPExtensions	List all registered NCP Extensions.
NWSendNCPExtensionRequest	Send a request to an NCP Extension. Have this function send the data from one location and place the reply in another.
NWSendNCPExtensionFraggedRequest	Send a request to an NCP Extension. Have this function gather the data from multiple addresses and place the reply in more than one address.

NCP Extension Tasks

12

This documentation describes common tasks associated with NCP Extension.

12.1 Accessing an NCP Extension from the Client

The client takes the following steps when accessing an NCP Extension:

1. Establish a connection with the server that has the NCP Extension registered.

```
   char serverUserCombo[96];
printf("\nEnter login [fileserver/]username to access echo
server:");
scanf("%s", serverUserCombo);
LoginToFileServer(serverUserCombo, 1, "");
```

   NCP Extension works through existing connections. This connection can be an attachment as well as a logged-in connection.

   Some of the functions that can be used to establish connections are:

   - AttachByAddress
   - AttachToFileServer
   - LoginToFileServer

2. Query to see if the desired NCP Extension has been registered.

```
   struct queryDataStruct {
      LONG CharsEchoed;
      LONG unused[7];
   } queryData;

   LONG NCPExtID;
   NWGetNCPExtensionInfo("ECHO SERVER", &NCPExtID, NULL, NULL,
NULL,
      &queryData);
   /* you should check the return value to see if the service
exists. */
   printf("ECHO SERVER reports %ld characters echoed so
far.\nKeystrokes"
      "will be echoed on the ECHO SERVER's screen, and echoed
locally\n"
      "after they are returned from the ECHO SERVER\n(Enter ctrl-z
to"
      "quit)\n",queryData.CharsEchoed);
   ...
}
```

   Before you can use the NCP Extension, it must be registered. To see if the extension has been registered, call NWGetNCPExtensionInfo, passing in the name of the NCP Extension you are looking for, such as ECHO SERVER. This function returns SUCCESSFUL if the NCP

Extension is registered and gives you an ID to use when accessing the NCP Extension. This ID is valid until the NCP Extension is deregistered. If no extension within a given ID is found then ERR_NO_ITEMS_FOUND will be returned. It may not be necessary to query in this fashion at all if a well known ID is being used.

The example above not only checks to see if the NCP Extension is registered, but it also uses the queryData pointer to receive information about how many characters the ECHO SERVER has echoed back to clients.

3 Access the NCP Extension.

```
main()
{
   LONG NCPExtID;
   char chr, rtnChr;
   LONG rtnSize;

   struct queryDataStruct {
      LONG CharsEchoed;
      LONG unused[7];
   } queryData;

   ...
   /* A connection would be established before making the following
call. */
   NWGetNCPExtensionInfo("ECHO SERVER", &NCPExtID, NULL, NULL,
NULL,
      &queryData);
   ...
   rtnSize = 1;
   while((chr = getch()) != CTRL_Z)   /* checking for ctrl-z to exit
*/
   {
      NWSendNCPExtensionRequest(NCPExtID, &chr, sizeof(chr),
&rtnChr,
                                &rtnSize);
      putchar(rtnChr);
   }
}
```

Your client can access an NCP Extension by first setting its thread group's current connection to the server with the NCP Extension, and then calling the NWSendNCPExtensionRequest, with the NCP Extension ID as one of the parameters. The example sends a request buffer with one character to the NCP Extension and receives a character in its reply buffer.

12.2 Providing an NLM Service as an NCP Extension

You must take the following steps to provide your NLM service as an NCP Extension.

1 Create your NCP Extension handler, connection event handler, and reply buffer manager functions, as well as a queryData update routine as needed (remember all the routines are optional).

```
BYTE EchoServer(NCPExtensionClient *client, BYTE *requestData,
   LONG requestDataLen, BYTE *replyData, LONG *replyDataLen)
{
   int savedThreadGroupID;
   savedThreadGroupID = GetThreadGroupID();
   SetThreadGroupID(myThreadGroupID);

   /* echo character */
   putchar(*(char *)requestData);
   *replyDataLen = 1;

   /* return echoed character */
   *replyData = *requestData;
   queryData->CharsEchoed++;
   SetThreadGroupID(savedThreadGroupID);
   return 0;
}
void EchoServerConnEventHandler(LONG connection, LONG eventType)
{
   ConsolePrintf("\nECHO SERVER notified of connection %d
                 logged out or returned\n", connection);
}
/* A buffer manager is not used in this example. */
```

In the example code above, EchoServer is the function that serves as the NCP Extension handler, and EchoServerConnDownHandler is called when certain connection events occurs. This example does not use a buffer manager.

You do not need to supply all of the routines. This example has a NCP Extension handler, a connection event handler, and a queryData update routine, but it does not use a reply buffer manager.

2 If needed, store the thread group ID so that it can be used for establishing CLIB context within your registered functions.

```
int myThreadGroupID;
main()
{
   myThreadGroupID = GetThreadGroupID();
   SetThreadContextSpecifier(GetThreadID(), NO_CONTEXT);
   ...
}
```

The functions registered for NCP Extension run as callbacks which run as OS threads. If these threads are going to call the NetWare API functions, you should manually give them CLIB context.

For the NetWare 4.x, 5.x, and 6.x OS, callback threads can be automatically given thread group context when they are registered. The context they are given is determined by the value of the registering thread's context specifier when the callbacks are registered. The context specifier can be set to give callbacks the thread group context of the calling thread, the thread group context of another thread group, or no context at all. When registering your callbacks for NCP Extension, it is recommended that you call SetThreadContextSpecifier with NO_CONTEXT as the `contextSpecifier` parameter so that the callbacks are not be automatically given context when they are registered. Then within your handler you would call

getThreadGroupID() with the appropriate thread group ID. This is recommended for performance and compatibility reasons.

For the NetWare 3.11 OS, threads do not have a context specifier, so you must manually set the context within each callback handler.

3 Register your NLM service as an NCP Extension.

```
struct queryDataStruct{
   LONG CharsEchoed;
   LONG unused[7];
} *queryData;
void main(void)
{
   ...
   NWRegisterNCPExtension("ECHO SERVER", EchoServer,\
      EchoServerConnDownHandler,NULL, 1, 0, 0, &queryData);
   queryData->CharsEchoed = 0;
   printf("Press any key to unload echo server.\n");
   getch();
   ...
}
```

An NLM can provide a service through NCP Extension by registering its service with the OS in one of the following ways:

- Calling NWRegisterNCPExtension to register the NCP Extension by using the name of the NCP Extension. This method returns a dynamic ID that is valid until the service providing NLM is unloaded.
- Calling NWRegisterNCPExtensionByID to register the NCP Extension using a well known ID that always associated with the NLM applications service.

After an NCP Extension has been registered, clients can access the NCP Extension. The Extension remains valid until the service-providing NLM deregisters the NCP Extension.

In this example, the NLM's service is registered with the server by calling NWRegisterNCPExtension.

The example above registers an extension handler with the name of "ECHO SERVER." EchoServer is the `NCPExtensionHandler`, EchoServerConnEventHandler is the `ConnectionEventHandler`, and NULL is passed in for `ReplyBufferManager`, meaning a reply buffer manager is not used. The `queryData` pointer becomes the handle to the NCP Extension.

4 Provide the service when your NCP Extension is accessed.

When the client requests service from your NCP Extension, the NetWare API first calls the function you registered in Step 3 as `NCPExtensionHandler`. This function is the workhorse that processes the request and fills a reply buffer that the NetWare API sends back to the client. After the buffer has been sent to the client, the NetWare API calls the function registered with `ReplyBufferManager`, if you have registered one. Remember if programming an NLM you must establish CLIB context as needed within all handlers.

The `ConnectionEventHandler` is called whenever a connection event occurs. Currently, notification occurs when connections are freed, killed, logged out, or restarted. This information is helpful for NCP Extensions that need to know when connection events occur. (These event types are discussed in Section 11.12, "Connection Status," on page 195.)

5 Deregister the NCP Extension.

```
struct queryDataStruct{
   LONG CharsEchoed;
   LONG unused[7];
} *queryData;

main()
{
   ...
   NWRegisterNCPExtension("ECHO SERVER", EchoServer,\
      EchoServerConnEventHandler,NULL, 1, 0, 0, &queryData);
   ...
   NWDeRegisterNCPExtension(queryData);
}
```

In most cases, you will choose to have your NLM provide its services as long as it is loaded. Before your NLM unloads, it must call NWDeRegisterNCPExtension to remove its NCP Extension from the list of NCP Extensions. If a NLM has more than one NCP Extension registered, it must call NWDeRegisterNCPExtension for each extension that it has registered.

NOTE: You cannot guarantee that outstanding NCP Extension requests complete successfully after NWDeRegisterNCPExtension is called.

If a client makes a call to an NCP Extension after the Extension has been deregistered, the client's call fails, returning ERR_NO_ITEMS_FOUND.

12.3 Registering Multiple NCP Extensions

There might be times when your service-providing NLM offers more than one service. If your NLM is a database, you may have the following services:

- Open the database
- Add a record
- Delete a record
- Search for a record
- Close the database

In the above case, you would have to make a decision: do you register five NCP Extensions to handle the requests, or do you register one NCP Extension that decodes a sub-function field within the request messages? If you choose to register five NCP Extensions, you must create five names for them. If you choose to use one NCP Extension, you only need to create one name (most NCPs operate this way).

If you choose to use a single NCP extension, your code might look like the following (see ncpscan.c.html (../../../samplecode/clib_sample/nlm/ncpscan/ncpscan.c.html)):

```
typedef MyStruct MyStruct;
struct requestDataStruct{
   int operation;
   char data[1000];
}MyStruct;
BYTE DataBaseControl(NCPExtensionClient *client,MyStruct *requestData,
    LONG requestDataLen, BYTE replyData, LONG *replyDataLen)
{
```

```
    switch(requestData->operation)
    {
    case OPEN_DATABASE:
       OpenDatabase(requestData->data);
       break;
    case ADD_RECORD:
       AddRecord(requestData->data);
       break;
    case DELETE_RECORD:
       DELETE_RECORD(requestData->data);
       break;
    case SEARCH_FOR_RECORD:
       SearchForRecord(requestData->data);
       break;
    case CLOSE_DATABASE:
       CloseDatabase(requestData->data);
    }
}
```

12.4 Allocating Reply Buffers

Reply buffers are allocated in the following ways:

- When a reply buffer manager is not used the NetWare API creates a single reply buffer and passes its address to the NCP Extension handler.
- The NCP Extension handler allocates a single reply buffer and returns a pointer to this buffer.
- The NCP Extension handler allocates multiple reply buffers and returns a pointer to a NCP extension message fragment structure that has pointers to the reply buffers.

NetWare API Allocation of a Single Reply Buffer: If your NCP Extension does not use a reply buffer manager, the NetWare API allocates a reply buffer that is the size specified by the client. The NetWare API then passes a pointer to the allocated buffer as a parameter into your NCP Extension handler. Your NCP Extension handler places its reply into the buffer, and the NetWare API sends the data in the buffer to the client.

NCP Extension Handler Allocation of a Single Reply Buffer: If your NCP Extension uses a reply buffer manager, the NetWare API does not allocate a reply buffer. In this case, it is the responsibility of the NCP Extension handler to provide the buffer. The NCP Extension handler places its data in the buffer it has allocated and then returns a pointer to the buffer. The NetWare API then sends the data in that buffer to the client.

NCP Extension Handler Allocation of Multiple Reply Buffers: If you wish to reply with data from multiple buffers, you may do so using a reply buffer manager.. In this case, the NCP Extension handler sets its `replyData` parameter to point to a structure containing pointers to multiple fragments. The NCP Extension handler also sets its `replyDataLen` parameter to REPLY_BUFFER_IS_FRAGGED. The NetWare API then sends the information from the multiple buffers.

The structure that you use to point to the fragmented data must be similar to the NCPExtensionMessageFrag structure that is documented in the reference for NWSendNCPExtensionFraggedRequest (page 238). The difference is that the structure your NCP Extension handler returns can have more than four elements in its `fragList`. (The client is limited

to four fragments, but there is no limit to the number of fragments that the NCP Extension handler can return.)

12.5 Processing an NCP Extension

When you register an NCP Extension with NWRegisterNCPExtension, three of the parameters are functions that may be called as part of the service. These parameters are `NCPExtensionHandler`, `ConnectionEventHandler`, and `ReplyBufferManager`. When the client calls the NCP Extension, the order of processing is in the following manner.

1. The client sends an NCP Extension request.

   If the client sends an NCP Extension request with NWSendNCPExtensionRequest the clients request must be contained in one buffer. If the client sends a request with NWSendNCPExtensionFraggedRequest, the client's request can be placed in one buffer or in multiple buffers depending on the client's architecture. Either way, the data is sent across the wire as a stream of bytes.

2. The NetWare API creates the needed buffer(s) on the server that is providing the NCP Extension.

   If a reply buffer manager has not been specified, the NetWare API creates two buffers on the server; the size of these buffers are the size of the client's request and reply buffers respectively.

   If a buffer manager has been specified, the NetWare API creates only one buffer the size of the client's request buffer. The creation of the reply buffer is the responsibility of the function registered as `NCPExtensionHandler`.

   If the request buffer is large, it is be sent in fragments to the server. The server reassembles the fragments, making the fragmentation transparent to your program.

3. The NetWare API calls the `NCPExtensionHandler` function.

   This workhorse function interprets the data in the request buffer and processes the request.

   If a reply buffer manager is not being used, this function places its reply data in the buffer that the NetWare API created. If a reply buffer manager is being used, this function returns a pointer to a buffer where it has stored the reply data.

4. The NetWare API sends the reply information to the client.

5. If the reply data is large, the NetWare API sends it across the wire in fragments. The client's NWSendNCPExtensionRequest or NWSendNCPExtensionFraggedRequest function reassembles the packet, making the fragmentation transparent to the user.

   NOTE: The data in the reply buffer is sent to the client only if the `NCPExtensionHandler` function returns SUCCESSFUL.

   ---

6. If a buffer manager was specified, the NetWare API calls the BufferManager.

   The buffer manager is called after that reply data has been sent to the client.

   In some ways, the reply buffer manager can be thought of as a second part of the NCP Extension handler. The reply buffer manager can free buffers and reset counters and semaphores that the NCP Extension handler has set. For example, if the NCP Extension handler has set a semaphore for a buffer, the buffer manager can signal or free the semaphore.

7. The NetWare API frees the buffers it has created.

When the NCP Extension request is completed, the NetWare API frees the buffers it has allocated for the request and reply data. When new requests come in, the NetWare API allocates new buffers.

NOTE: The `ConnectionEventHandler` is currently only called when a connection is freed, killed, logged out, or restarted.

12.6 Deregistering Before Unloading

Before an NLM unloads, it must deregister all NCP Extensions that it has registered. Failure to deregister before unloading can cause the server to abend.

When an NCP Extension is deregistered, all new requests return with ERR_NO_ITEMS_FOUND, and existing requests may or may not be completed. Those that don't complete also return with the value of ERR_NO_ITEMS_FOUND.

If you need to be assured that all of your current requests are completed, you can set a counter telling how many requests are outstanding. Outstanding requests are requests being processed by the extension handler or the reply buffer manager. You decrement the counter when a request has completed. Before deregistering the NCP Extension, you return a failure return code immediately for all new requests and continue servicing the current requests. When the counter is set to zero, you call NWDeRegisterNCPExtension, then continue letting your NLM unload.

See ncpscan.c.html (../../../samplecode/clib_sample/nlm/ncpscan/ncpscan.c.html).

NCP Extension Functions

13

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

NWDeRegisterNCPExtension

Deregisters an NCP Extension

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM

Service: NCP Extension

Syntax

```
#include <nlm\nwncpx.h>

int NWDeRegisterNCPExtension  (
   void  *queryData);
```

Parameters

queryData

(IN) Points to the extension handle received from the NWRegisterNCPExtension function.

Return Values

The following table lists return values and descriptions.

Value	Hex	Name and description
0	0x00	SUCCESSFUL The extension was deregistered
255	0xFF	ERR_NO_ITEMS_FOUND The extension has already been deregistered

Remarks

NWDeRegisterNCPExtension is called by the service-providing NLM applications in conjunction with the NWRegisterNCPExtension function.

When an NCP Extension is registered with the NWRegisterNCPExtension function, the address of the `queryData` parameter is passed as one of the parameters. The pointer is then initialized to point to a 32-byte area of memory in which the service provider can place data. The `queryData` parameter is also used as a handle for deregistering the NCP Extension.

See Also

NWGetNCPExtensionInfo (NLM) (page 211), NWRegisterNCPExtension (page 226), NWScanNCPExtensions (page 233), NWSendNCPExtensionRequest (page 240)

NWFragNCPExtensionRequest

Sends and receives information from an NCP extension handle

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

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

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

Service: NCP Extension

Syntax

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

NWCCODE N_API NWFragNCPExtensionRequest (
   NWCONN_HANDLE       conn,
   nuint32             NCPExtensionID,
   nuint16             reqFragCount,
   NW_FRAGMENT N_FAR  *reqFragList,
   nuint16             replyFragCount,
   NW_FRAGMENT N_FAR  *replyFragList);
```

Delphi Syntax

```
uses calwin32

Function NWFragNCPExtensionRequest
  (conn : NWCONN_HANDLE;
   NCPExtensionID : nuint32;
   reqFragCount : nuint16;
   Var reqFragList : NW_FRAGMENT;
   replyFragCount : nuint16;
   Var replyFragList : NW_FRAGMENT
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

NCPExtensionID

(IN) Specifies the ID of the NCP extension handler to use for the request.

reqFragCount

(IN) Specifies the number of request fragments.

reqFragList

(IN) Points to the NW_FRAGMENT structure.

replyFragCount

(IN) Specifies the number of reply fragments.

replyFragList

(IN/OUT) Points to the NW_FRAGMENT structure.

Remarks

The fragment based protocol allows data up to 64K (a server imposed limitation) to be transferred to and from the NCP extension handler.

To increase packet efficiency, NWFragNCPExtensionRequest packs as many fragments as possible into a send buffer.

The reply data will be returned in the NW_FRAGMENT structure pointed to by the `replyFragList` parameter. The `fragSize` field of the NW_FRAGMENT structure will be updated to reflect the number of bytes copied into the buffer pointed to by the `fragAddress` field.

NCP Calls

0x2222 23 17 Get File Server Information

NWGetNCPExtensionInfo

Returns information about the specified NCP extension handler

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: Windows NT

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

Service: NCP Extension

Syntax

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

NWCCODE N_API  NWGetNCPExtensionInfo (
   NWCONN_HANDLE   conn,
   nuint32         NCPExtensionID,
   pnstr8          NCPExtensionName,
   pnuint8         majorVersion,
   pnuint8         minorVersion,
   pnuint8         revision,
   pnuint8         queryData);
```

Delphi Syntax

```
uses calwin32

Function NWGetNCPExtensionInfo
  (conn : NWCONN_HANDLE;
   NCPExtensionID : nuint32;
   NCPExtensionName : pnstr8;
   majorVersion : pnuint8;
   minorVersion : pnuint8;
   revision : pnuint8;
   queryData : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

NCPExtensionID

(IN) Specifies the ID of the NCP extension handler for which to get information.

NCPExtensionName

(OUT) Points to a buffer to receive NCP extension name (33 bytes, optional).

majorVersion

(OUT) Points to the major version number of the NCP extension handler (optional).

minorVersion

(OUT) Points to the minor version number of the NCP extension handler (optional).

revision

(OUT) Points to the revision number of the NCP extension handler (optional).

queryData

(OUT) Points to a 32-byte buffer of custom information that the NCP extension handler can use (optional).

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	SUCCESSFUL The extension was found, and the non-null output parameters were filled.
0x89FE	Extension ID not found

Remarks

If an NCP extension with an ID higher than the one submitted was found, and its data was returned, NWGetNCPExtensionInfo returns 0x89FE.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 36 00 Scan NCP Extensions
0x2222 36 02 Scan Loaded Extensions By Name
0x2222 36 05 Get NCP Extension Info

See Also

NWNCPExtensionRequest (page 223), NWFragNCPExtensionRequest (page 207), NWScanNCPExtensions (page 233), NWGetNCPExtensionInfoByName (page 217), NWGetNCPExtensionsList (page 219), NWGetNumberNCPExtensions (page 221)

NWGetNCPExtensionInfo (NLM)

Returns information about an NCP Extension specified by name

Local Servers: nonblocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM

Service: NCP Extension

Syntax

```
#include <nlm\nwncpx.h>

int NWGetNCPExtensionInfo  (
   const char  *NCPExtensionName,
   LONG        *NCPExtensionID,
   BYTE        *majorVersion,
   BYTE        *minorVersion,
   BYTE        *revision,
   void          *queryData);
```

Parameters

NCPExtensionName

(IN) Points to the name of the desired NCP Extension.

NCPExtensionID

(OUT) Points to the ID of the desired NCP Extension (optional).

majorVersion

(OUT) Points to the major version number of the NCP Extension provider (optional).

minorVersion

(OUT) Points to the minor version number of the NCP Extension provider (optional).

revision

(OUT) Points to the revision number of the NCP Extension provider (optional).

queryData

(OUT) Points to 32 bytes of information from the NCP Extension (optional).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name and description
0	0x00	SUCCESSFUL The extension was found, and the non-null output parameters were filled.
255	0xFF	ERR_NO_ITEMS_FOUND The extension name was not found.
1-16		A communications error occurred. (See nwncpext.h.)

Remarks

NWGetNCPExtensionInfo (NLM), the NWScanNCPExtensions (NLM) function, and the NWSendNCPExtensionRequest function access NCP Extensions. For example, if you know the name of the NCP Extension you want to access, such as "ECHO SERVER," you can call NWGetNCPExtensionInfo (NLM) for the following purposes:

- To see if the NCP Extension is registered.
- To check the version of the NCP Extension.
- To get the NCP Extension ID number used when calling the NWSendNCPExtensionRequest function.
- To receive 32 bytes of information from the NCP Extension without calling the NWSendNCPExtensionRequest function.

Before a client can access an NCP Extension, NWGetNCPExtensionInfo (NLM) must be called to see if the Extension has been registered followed by calling the NWScanNCPExtensions (NLM) function to receive the Extension ID needed to call the NCP Extension. If the NCP Extension has been registered, NWGetNCPExtensionInfo (NLM) returns SUCCESSFUL; otherwise, it returns ERR_NO_ITEMS_FOUND. The NWScanNCPExtensions (NLM) function returns the same information but must be called iteratively until the NCP Extension name is found.

The `NCPExtensionName` parameter can be any character string, up to 32 bytes plus a NULL terminator. The NCP Extension names are case sensitive and must be unique for each NCP Extension. One suggestion is to name the NCP Extension the same as your NLM. To avoid naming conflicts, you should clear your NCP Extension's name through Developer Support.

You provide the `majorVersion`, `minorVersion`, and `revision` parameters when you call the NWRegisterNCPExtension function. If you have different versions or revisions of the NCP Extension, the client can use these parameters to verify that the extension is the correct version. If you do not want to use any of these parameters, pass NULL.

The server side and the client side of NCP Extensions should be implemented as matched sets so the client side knows what the server side is expecting and what it can return. The client side also needs to know the name of the NCP Extension.

There are some cases where NWGetNCPExtensionInfo (NLM) can return all of the information your client needs, eliminating the need to call the NWSendNCPExtensionRequest function or to have an NCP Extension handler. This information is placed in the client's `queryData` buffer, whose address is passed as a parameter to NWGetNCPExtensionInfo (NLM).

Use this method if the service-providing NLM is periodically updating its `queryData` buffer (with 32 bytes or less of information) and the buffer address was returned to the NLM when it called the NWRegisterNCPExtension function. If the information you want is in the NLM's `queryData`

buffer, call NWGetNCPExtensionInfo (NLM) to copy the contents of the `queryData` buffer for the service-providing NLM into the `queryData` buffer for the client. This method is useful only if a one-way server-to-client message is sufficient.

If you are using the `queryData` buffer, pass NULL to the `queryData` parameter.

NOTE: If an NLM is unloaded, all NCP Extensions associated with it are deregistered. If the NLM is reloaded, its NCP Extensions do not have the same NCP Extension IDs, even though they have the same names.

If any of the client (NLM or workstation) NCP Extension functions return ERR_NO_ITEMS_FOUND (or ERR_NCPEXT_NO_HANDLER after previously working properly), call the NCPGetExtensionInfo function again. The NCPGetExtensionInfo function will return the new `NCPExtensionID` parameter if the NCP Extension has been deregistered and then reregistered.

See Also

NWDeRegisterNCPExtension (page 206), NWGetNCPExtensionInfoByID (page 214), NWRegisterNCPExtension (page 226), NWScanNCPExtensions (NLM) (page 235), NWSendNCPExtensionRequest (page 240)

NWGetNCPExtensionInfoByID

Returns information about an NCP Extension specified by ID

Local Servers: nonblocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM

Service: NCP Extension

Syntax

```
#include <nlm\nwncpx.h>

int NWGetNCPExtensionInfoByID  (
   LONG   NCPExtensionID,
   char  *NCPExtensionName,
   BYTE  *majorVersion,
   BYTE  *minorVersion,
   BYTE  *revision,
   void  *queryData);
```

Parameters

NCPExtensionID

(IN) Specifies the ID of the desired NCP Extension.

NCPExtensionName

(OUT) Points to the name of NCP Extension associated with the ID passed in the `NCPExtensionID` parameter (optional).

majorVersion

(OUT) Points to the major version number of the NCP Extension provider (optional).

minorVersion

(OUT) Points to the minor version number of the NCP Extension provider (optional).

revision

(OUT) Points to the revision number of the NCP Extension provider (optional).

queryData

(OUT) Points to 32 bytes of information from the NCP Extension (optional).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name and description
0	0x00	SUCCESSFUL The extension was found, and the non-null output parameters were filled.
255	0xFF	ERR_NO_ITEMS_FOUND The extension name was not found.
1-16		A communications error occurred. (See niterror.h.)

Remarks

The NWGetNCPExtensionInfo (NLM), NWGetNCPExtensionInfoByID, NWScanNCPExtensions (NLM), NWSendNCPExtensionFraggedRequest, and NWSendNCPExtensionRequest functions access NCP Extensions.

If you know the ID of the NCP Extension, you can call NWGetNCPExtensionInfoByID for the following purposes:

- To see if the NCP Extension is registered.
- To verify the name of the NCP Extension.
- To check the version of the NCP Extension handler.
- To receive 32 bytes of information from the NCP Extension without calling the NWSendNCPExtensionRequest function.

Before a client can access an NCP Extension, call either the NWGetNCPExtensionInfo (NLM), NWGetNCPExtensionInfoByID, or NWScanNCPExtensions (NLM) function to see if the extension has been registered. If the NCP Extension has been registered, NWGetNCPExtensionInfoByID returns SUCCESSFUL; otherwise, it returns ERR_NO_ITEMS_FOUND. The NWGetNCPExtensionInfo (NLM) and NWScanNCPExtensions (NLM) functions return the same information but they use the name of the NCP Extension, rather than the ID.

The `NCPExtensionID` parameter can be a dynamic ID returned from the NWGetNCPExtensionInfo (NLM) or NWScanNCPExtensions (NLM) function, or it can be a static ID assigned by Developer Support.

If you are using a static ID, check the name pointed to by the `NCPExtensionName` parameter (on the first call) to verify that the name returned is the same as the name of your NCP Extension.

The `majorVersion`, `minorVersion`, and `revision` parameters are those you provide when you call the NWRegisterNCPExtension or NWRegisterNCPExtensionByID function. If you have different versions or revisions of the NCP Extension service providers, the client can use these parameters to verify that the service provider is the correct version. If you do not want to use any of these parameters, pass NULL.

There are some cases where NWGetNCPExtensionInfoByID can return all of the information your client needs, eliminating the need to call the NWSendNCPExtensionRequest function or to have an NCP Extension handler. This information is placed in the client's `queryData` buffer, whose address is passed as a parameter to NWGetNCPExtensionInfoByID.

Use this method if the service-providing NLM is periodically updating its `queryData` buffer (with 32 bytes or less of information) and whose address was returned to the NLM when it called NWRegisterNCPExtension or NWRegisterNCPExtensionByID. If the information you want is in

the NLM's `queryData` buffer, you can use NWGetNCPExtensionInfoByID to copy the contents of the `queryData` buffer for the service-providing NLM into the `queryData` buffer for the client. This method is useful only if a one-way server-to-client message is sufficient.

If you are using the `queryData` buffer, pass NULL to the `queryData` parameter.

NOTE: If an NLM is unloaded, all NCP Extensions associated with it are deregistered. If the NLM is reloaded, and it registers its NCP Extensions by calling NWRegisterNCPExtensionByID, the IDs for the extensions are the same.

If the NLM is reloaded, and it registers its NCP Extensions by name by calling the NWRegisterNCPExtension function, the NCP Extensions do not have the same NCP Extension IDs, even though they have the same names.

If any of the client (NLM or workstation) NCP Extension functions return ERR_NO_ITEMS_FOUND (or ERR_NCPEXT_NO_HANDLER after previously working properly), call the NWGetNCPExtensionInfo (NLM) function. The NWGetNCPExtensionInfo (NLM) function will return the new `NCPExtensionID` parameter if the NCP Extension has been deregistered and then reregistered.

See Also

NWDeRegisterNCPExtension (page 206), NWGetNCPExtensionInfo (NLM) (page 211), NWRegisterNCPExtension (page 226), NWScanNCPExtensions (NLM) (page 235), NWSendNCPExtensionRequest (page 240)

NWGetNCPExtensionInfoByName

Returns information for the specified NCP extension handler

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT

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

Service: NCP Extension

Syntax

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

NWCCODE N_API NWGetNCPExtensionInfoByName (
   NWCONN_HANDLE       conn,
   const nstr8 N_FAR  *NCPExtensionName,
   pnuint32            NCPExtensionID,
   pnuint8             majorVersion,
   pnuint8             minorVersion,
   pnuint8             revision,
   pnuint8             queryData);
```

Delphi Syntax

```
uses calwin32

Function NWGetNCPExtensionInfoByName
  (conn : NWCONN_HANDLE;
   NCPExtensionName : pnstr8;
   NCPExtensionID : pnuint32;
   majorVersion : pnuint8;
   minorVersion : pnuint8;
   revision : pnuint8;
   queryData : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

NCPExtensionName

(IN) Points to a buffer containing the NCP extension name (33 bytes) for which to get information (optional).

NCPExtensionID

(OUT) Points to the ID of the NCP extension handler.

majorVersion

(OUT) Points to the major version number of the NCP extension handler (optional).

minorVersion

(OUT) Points to the minor version number of the NCP extension handler (optional).

revision

(OUT) Points to the revision number of the NCP extension handler (optional).

queryData

(OUT) Points to a 32-byte buffer of custom information the NCP extension handler can optionally use (optional).

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 36 00 Scan NCP Extensions
0x2222 36 02 Scan Currently Loaded NCP Extensions By Name

See Also

NWGetNCPExtensionInfo (page 209), NWNCPExtensionRequest (page 223), NWFragNCPExtensionRequest (page 207), NWScanNCPExtensions (page 233), NWGetNCPExtensionsList (page 219), NWGetNumberNCPExtensions (page 221)

NWGetNCPExtensionsList

Returns a list of NCP extension handlers loaded on the server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT

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

Service: NCP Extension

Syntax

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

NWCCODE N_API  NWGetNCPExtensionsList (
   NWCONN_HANDLE   conn,
   pnuint32        startNCPExtensionID,
   pnuint16        itemsInList,
   pnuint32        NCPExtensionIDList);
```

Delphi Syntax

```
uses calwin32

Function NWGetNCPExtensionsList
  (conn : NWCONN_HANDLE;
   startNCPExtensionID : pnuint32;
   itemsInList : pnuint16;
   NCPExtensionIDList : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNCPExtensionID

(IN/OUT) Points to the next extension ID to use to obtain a list.

itemsInList

(OUT) Points to the number of NCP extension handler IDs.

NCPExtensionIDList

(OUT) Points to a buffer to receive list of NCP extension handler IDs (512 bytes or 4 times the number of NCP extension IDs, whichever is less).

Remarks

If there are more than 128 extension handlers loaded, call NWGetNCPExtensionsList multiple times.

Set `startNCPExtensionID` to 0 for the first iteration. NWGetNCPExtensionsList returns the next value to use.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 36 0 Scan Loaded NCP Extensions
0x2222 36 04 Get NCP Extension Loaded List

See Also

NWGetNCPExtensionInfo (page 209), NWNCPExtensionRequest (page 223), NWFragNCPExtensionRequest (page 207), NWScanNCPExtensions (page 233), NWGetNCPExtensionInfoByName (page 217), NWGetNumberNCPExtensions (page 221)

NWGetNumberNCPExtensions

Returns the number of NCP extension handlers loaded on the specified server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT

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

Service: NCP Extension

Syntax

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

NWCCODE N_API  NWGetNumberNCPExtensions (
   NWCONN_HANDLE   conn,
   pnuint32        numNCPExtensions);
```

Delphi Syntax

```
uses calwin32

Function NWGetNumberNCPExtensions
  (conn : NWCONN_HANDLE;
   numNCPExtensions : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

numNCPExtensions

(OUT) Points to the number of NCP extension handlers installed on the server.

NCP Calls

0x2222 23 17 Get Server Info
0x2222 36 0 Scan Loaded NCP Extensions
0x2222 36 3 Get Number Of Loaded NCP Extensions

See Also

NWGetNCPExtensionInfo (page 209), NWNCPExtensionRequest (page 223), NWFragNCPExtensionRequest (page 207), NWScanNCPExtensions (page 233), NWGetNCPExtensionInfoByName (page 217), NWGetNCPExtensionsList (page 219)

NWNCPExtensionRequest

Sends and receives small data from an NCP extension handler

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT

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

Service: NCP Extension

Syntax

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

NWCCODE N_API NWNCPExtensionRequest (
   NWCONN_HANDLE      conn,
   nuint32            NCPExtensionID,
   const void N_FAR  *requestData,
   nuint16            requestDataLen,
   void N_FAR        *replyData,
   pnuint16           replyDataLen);
```

Delphi Syntax

```
uses calwin32

Function NWNCPExtensionRequest
  (conn : NWCONN_HANDLE;
   NCPExtensionID : nuint32;
   const requestData : nptr;
   requestDataLen : nuint16;
   replyData : nptr;
   replyDataLen : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

NCPExtensionID

(IN) Specifies the ID of the NCP extension handler to use for the request.

requestData

(IN) Points to a buffer containing request data.

requestDataLen

(IN) Specifies the length of request data.

replyData

(OUT) Points to a buffer to receive reply data (can be the same buffer as request data; optional if no reply data is expected).

replyDataLen

(IN/OUT) Points to amount of data expected and how much data was returned (optional if no reply data is expected).

Remarks

NWNCPExtensionRequest will take any size data buffer and send it to the server. Requests larger than 500 bytes will be processed by calling NWFragNCPExtensionRequest which breaks up the data and sends it in multiple packets.

NCP Calls

0x2222 23 17 Get File Server Information

See Also

NWGetNCPExtensionInfo (page 209), NWFragNCPExtensionRequest (page 207), NWScanNCPExtensions (page 233), NWGetNCPExtensionInfoByName (page 217), NWGetNCPExtensionsList (page 219), NWGetNumberNCPExtensions (page 221)

NWNCPSend

Sends an NCP request to a currently connected server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM

Service: NCP Extension

Syntax

```
#include <nlm\nwconn.h>

int NWNCPSend  (
   BYTE         functionCode,
   const void  *sendPacket,
   WORD         sendLen,
   void        *replyBuf,
   WORD           replyLen);
```

Parameters

`functionCode`

(IN) Specifies the NCP function code.

`sendPacket`

(IN) Points to the input buffer for the NCP.

`sendLen`

(IN) Specifies the length of the `sendPacket` parameter.

`replyBuf`

(IN/OUT) Points to the reply buffer for the NCP.

`replyLen`

(IN/OUT) Specifies the length of the `replyBuf` parameter.

Return Values

ESUCCESS or NetWare errors.

Remarks

An NCP request consists of function code and a request buffer that contains input information needed to process the request.

NWRegisterNCPExtension

Registers a service to be provided as an NCP extension

Local Servers: blocking

Remote Servers: N/A

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM

Service: NCP Extension

Syntax

```
#include <nlm\nwncpx.h>

int NWRegisterNCPExtension  (
   const char    *NCPExtensionName,
   BYTE         (*NCPExtensionHandler)(
      NCPExtensionClient   *client,
      void                * requestData,
      LONG                 requestDataLen,
      void                * replyData,
      LONG                *replyDataLen),
   void         (*ConnectionEventHandler)(
      LONG   connection,
      LONG   eventType)
   void         (*ReplyBufferManager)(
      NCPExtensionClient   *client,
      void                * replyBuffer),
   BYTE          majorVersion,
   BYTE          minorVersion,
   BYTE          revision,
   void        **queryData);
```

Parameters

NCPExtensionName

(IN) Points to the name of an NCP Extension.

NCPExtensionHandler

(IN) Points to the function to be called when the NCP Extension calls the NWSendNCPExtensionRequest function (optional).

ConnectionEventHandler

(IN) Points to the function to be called and action to follow when a connection is freed, killed, logged out, or restarted (optional).

ReplyBufferManager

(IN) Points to a buffer manager function used to reply to NCP Extension requests (optional).

majorVersion

(IN) Specifies the major version number of the service provider.

minorVersion

(IN) Specifies the minor version number of the service provider.

revision

(IN) Specifies the revision number of the service provider.

queryData

(OUT) Points to a 32-byte area that NetWare has allocated.

Return Values

The following table lists return values and descriptions.

Value	Hex	Name and description
0	0x00	SUCCESSFUL The extension was found, and the non-null output parameters were filled.
5	0x05	ENOMEM Not enough memory was available on the server to register the service.
166	0xA6	ERR_ALREADY_IN_USE The NCP Extension name is already registered. Your service is not registered.
255	0xFF	ERR_BAD_PARAMETER The `NCPExtensionName` parameter is longer than the 32-byte limit.

Remarks

NWRegisterNCPExtension is called by the service-providing NLM applications in conjunction with the NWDeRegisterNCPExtension function.

NCP extension names are case sensitive and must be unique. They have a maximum length of 32 bytes plus a NULL terminator.

The `queryData` parameter can be used by the service provider to return up to 32 bytes of information to the client and is aligned on a DWORD (32-bit) boundary. This information can then be retrieved by calling the NWGetNCPExtensionInfo (NLM) or NWScanNCPExtensions (NLM) function. The `queryData` parameter is also used by the registering NLM as the NCP extension handle when the NWDeRegisterNCPExtension function is called.

NOTE: The `NCPExtensionHandler` parameter returns a BYTE representing the value returned when the NWSendNCPExtensionRequest function is called. The extension handler can return any value other than those used by the lower-level NCP-transport services (see niterror.h). However, information is placed into the `replyData` parameter after the `NCPExtensionHandler` parameter returns SUCCESSFUL.

Other status information can be returned to the client with the extension handler. However, do not return any values (other than SUCCESSFUL) that NWRegisterNCPExtension can return. Otherwise, future versions of the OS might return values you have defined and confuse their

meaning. If the extension handler always returns SUCCESSFUL and then uses a "status" field in the `replyData` parameter to return status information, the meaning of each return value will be clear.

If you can provide all needed information by updating the 32-byte `queryData` buffer, pass NULL to the `NCPExtensionHandler` parameter. Then call either the NWGetNCPExtensionInfo (NLM) or NWScanNCPExtensions (NLM) function to obtain information in the `queryData` buffer. This is a passive method of passing information. The NCP extension will not be notified that the `queryData` parameter was accessed.

NOTE: The `NCPExtensionHandler`, `ConnectionEventHandler`, and `ReplyBufferManager` parameters are function callbacks that run as OS threads. They need to have CLIB context if they are going to make calls into CLIB that need context.

The function pointed to by the `NCPExtensionHandler` parameter has the following parameters:

`client`

(IN) Points to the NCPExtensionClient (page 245) structure containing the connection and task of the calling client (also used by the `ReplyBufferManager` parameter to associate the request with the reply notification it receives).

`requestData`

(IN) Points to a buffer, which might be DWORD aligned, to hold the request information.

`requestDataLen`

(IN) Specifies the size (in bytes) of the data in the `requestData` parameter.

`replyData`

(OUT) Points to a buffer to store the response data from the service routine if the `ReplyBufferManager` parameter is NULL. Otherwise, points to the address of a valid buffer, which might be DWORD aligned, that the NCP extension handler created.

`replyDataLen`

(IN/OUT) Inputs the maximum size (in bytes) of information that can be stored in the reply buffer. Outputs the actual number of bytes that the `NCPExtensionHandler` parameter stored in the reply buffer.

The function pointed to by the `ConnectionEventHandler` parameter has the following parameters:

`connection`

(IN) Specifies the connection number for any connection (NCP extension clients and others) that was logged out or cleared (optional).

`eventType`

(IN) Specifies the type of event that is being reported for NetWare 3.12 and higher (optional):

CONNECTION_BEING_FREED
CONNECTION_BEING_KILLED
CONNECTION_BEING_LOGGED_OUT
CONNECTION_BEING_FREED

You must decide if it is important for your service to be aware of when clients (particularly the NCP extension clients) log out or terminate a connection.

The `ConnectionEventHandler` parameter does not return a value.

The function pointed to by the `ReplyBufferManager` parameter has the following parameters:

`client`

(IN) Points to the NCPExtensionClient (page 245) structure containing the connection and task of the calling client.

`replyBuffer`

(IN) Points to a buffer whose information has been returned to the client (optional).

See Also

GetThreadContextSpecifier (*NDK: NLM Threads Management*), NWDeRegisterNCPExtension (page 206), NWGetNCPExtensionInfo (NLM) (page 211), NWScanNCPExtensions (NLM) (page 235), NWSendNCPExtensionRequest (page 240), NWRegisterNCPExtensionByID (page 230), SetThreadContextSpecifier (*NDK: NLM Threads Management*)

NWRegisterNCPExtensionByID

Registers a service to be provided as an NCP Extension and assigns the NCP Extension a specific ID

Local Servers: blocking

Remote Servers: N/A

NetWare Server: 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM

Service: NCP Extension

Syntax

```
#include <nlm\nwncpx.h>

int NWRegisterNCPExtensionByID  (
   LONG          NCPExtensionID,
   const char  *NCPExtensionName,
   BYTE       (*NCPExtensionHandler)(
      NCPExtensionClient   *NCPExtensionClient,
      void                * requestData,
      LONG                 requestDataLen,
      void                * replyData,
      LONG                *replyDataLen),
   void       (*ConnectionEventHandler)(
      LONG   connection,
      LONG   eventType)
   void        (*ReplyBufferManager)(
      NCPExtensionClient   *NCPExtensionClient,
      void                * replyBuffer),
   BYTE        majorVersion,
   BYTE        minorVersion,
   BYTE        revision,
   void      **queryData);
```

Parameters

NCPExtensionID

(IN) Specifies the unique ID to be associated with your service for the NCP Extension (assigned by Developer Support).

NCPExtensionName

(IN) Points to the name to identify the NCP Extension.

NCPExtensionHandler

(IN) Points to the function to be called when the NCP Extension calls the NWSendNCPExtensionRequest or NWSendNCPExtensionFraggedRequest function (optional).

ConnectionEventHandler

(IN) Points to the function to be called and steps to follow when a connection is freed, killed, logged out, or restarted (optional).

ReplyBufferManager

(IN) Points to a buffer manager function used to reply to NCP Extension requests (optional).

majorVersion

(IN) Specifies the major version number of the service provider.

minorVersion

(IN) Specifies the minor version number of the service provider.

revision

(IN) Specifies the revision number of the service provider.

queryData

(OUT) Points to a 32-byte area that NetWare has allocated.

Return Values

The following table lists return values and descriptions.

Value	Hex	Name and description
0	0x00	SUCCESSFUL The extension was found, and the non-null output parameters were filled.
5	0x05	ENOMEM Not enough memory was available on the server to register the service.
166	0xA6	ERR_ALREADY_IN_USE The NCP Extension name is already registered. Your service is not registered.
251	0xFB	ERR_UNKNOWN_REQUEST The server version does not support this request.
255	0xFF	ERR_BAD_PARAMETER The `NCPExtensionName` parameter is longer than the 32-byte limit.

Remarks

NWRegisterNCPExtensionByID is called by the service-providing NLM applications in conjunction with NWDeRegisterNCPExtension and NWRegisterNCPExtension.

NCP extension names are case sensitive and must be unique. They have a maximum length of 32 bytes plus a NULL terminator.

For an explanation of the `NCPExtensionHandler`, `ConnectionEventHandler`, and `ReplyBufferManager` parameters, see the Remarks section for NWRegisterNCPExtension (page 226).

The `queryData` parameter can be used by the service provider to return up to 32 bytes of information to the client and is aligned on a DWORD (32-bit) boundary. This information can then be retrieved by calling NWGetNCPExtensionInfo (NLM), NWGetNCPExtensionInfoByID, or

NWScanNCPExtensions (NLM). The `queryData` parameter is also used by the registering NLM as the NCP extension handle when NWDeRegisterNCPExtension is called.

NOTE: The `NCPExtensionHandler`, `ConnectionEventHandler`, and `ReplyBufferManager` parameters are function callbacks that run as OS threads. They need to have CLIB context if they are going to make calls into CLIB that need context.

See Also

GetThreadContextSpecifier (*NDK: NLM Threads Management*), NWDeRegisterNCPExtension (page 206), NWRegisterNCPExtension (page 226), SetThreadContextSpecifier (*NDK: NLM Threads Management*)

NWScanNCPExtensions

Scans the server for NCP extension handlers

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

Platform: Windows NT

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

Service: NCP Extension

Syntax

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

NWCCODE N_API  NWScanNCPExtensions (
   NWCONN_HANDLE   conn,
   pnuint32        NCPExtensionID,
   pnstr8          NCPExtensionName,
   pnuint8         majorVersion,
   pnuint8         minorVersion,
   pnuint8         revision,
   pnuint8         queryData);
```

Delphi Syntax

```
uses calwin32

Function NWScanNCPExtensions
  (conn : NWCONN_HANDLE;
   NCPExtensionID : pnuint32;
   NCPExtensionName : pnstr8;
   majorVersion : pnuint8;
   minorVersion : pnuint8;
   revision : pnuint8;
   queryData : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

NCPExtensionID

(IN/OUT) Points to the ID of the NCP extension handler for which to get information (set to -1 initially).

NCPExtensionName

(OUT) Points to the 32-byte buffer to receive the NCP extension name (optional).

majorVersion

(OUT) Points to the major version number of the NCP extension handler (optional).

minorVersion

(OUT) Points to the minor version number of the NCP extension handler (optional).

revision

(OUT) Points to the revision number of the NCP extension handler (optional).

queryData

(OUT) Points to the 32-byte buffer of custom information the NCP extension handler can use (optional).

NCP Calls

0x2222 36 00 Scan Currently Loaded NCP Extensions

See Also

NWGetNCPExtensionInfo (page 209), NWNCPExtensionRequest (page 223), NWFragNCPExtensionRequest (page 207), NWGetNCPExtensionInfoByName (page 217), NWGetNCPExtensionsList (page 219), NWGetNumberNCPExtensions (page 221)

NWScanNCPExtensions (NLM)

Iteratively returns information about all registered NCP extensions

Local Servers: nonblocking

Remote Servers: blocking

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

Platform: NLM

Service: NCP Extension

Syntax

```
#include <nlm\nwncpx.h>

int NWScanNCPExtensions  (
   LONG  *NCPExtensionID,
   char  *NCPExtensionName,
   BYTE  *majorVersion,
   BYTE  *minorVersion,
   BYTE  *revision,
   void  *queryData);
```

Parameters

NCPExtensionID

(IN/OUT) Points to the ID of the desired NCP Extension.

NCPExtensionName

(OUT) Points to the name of the NCP Extension that was found.

majorVersion

(OUT) Points to the major version number of the NCP Extension provider (optional).

minorVersion

(OUT) Points to the minor version number of the NCP Extension provider (optional).

revision

(OUT) Points to the revision number of the NCP Extension provider (optional).

queryData

(OUT) Points to 32 bytes of information from the NCP Extension service provider and allocated when the NLM calls the NWRegisterNCPExtension function (optional).

Return Values

The following table lists return values and descriptions.

0	SUCCESSFUL: Extension was found and non-NULL output parameters were filled
-1	No more NCP Extensions were found
1-16	An NCP error occurred (see niterror.h)

Remarks

NWScanNCPExtensions (NLM) can be used iteratively to return the names of all the NCP Extensions registered on the server being queried. To scan the complete list of NCP Extensions, set the `NCPExtensionID` parameter to BEGIN_SCAN_NCP_EXTENSIONS. When NWScanNCPExtensions (NLM) returns, the `NCPExtensionID` parameter will be set to the ID of the first NCP Extension in the list and will return SUCCESSFUL. Use the ID in the `NCPExtensionID` parameter as a seed value to find the next NCP Extension ID. Continue calling NWScanNCPExtensions (NLM), using the new IDs returned in the `NCPExtensionID` parameter, until you find the information you want or until -1 is returned.

Call NWScanNCPExtensions (NLM) when you want to list the names of the NCP Extensions but are not looking for the name of a specific extension. If you know the name of your NCP Extension, such as "My NCP Extension," you should call NWGetNCPExtensionInfo (NLM) to see if the extension is registered because NWGetNCPExtensionInfo (NLM) needs to be called only once.

Call NWScanNCPExtensions (NLM) to do the following:

- See if the NCP Extension is registered.
- Check the version of the NCP Extension.
- Get the NCP Extension ID number to call NWSendNCPExtensionRequest.
- Receive 32 bytes of information from the NCP Extension without calling the NWSendNCPExtensionRequest function.

The `NCPExtensionName` parameter should be set to a buffer that is MAX_NCP_EXTENSION_NAME_BYTES (33) bytes long. The returned name is case sensitive and unique for each NCP Extension.

NOTE: The IDs of NCP Extensions are not always consecutive numbers. Therefore, you should not assume that if you increment the value in the `NCPExtensionID` parameter by one that it is a valid NCP Extension ID.

The `majorVersion`, `minorVersion`, and `revision` parameters are assigned by the registering NLM when it calls NWRegisterNCPExtension. If you have different versions or revisions of the NCP Extension, use these fields to verify that the extension is of the correct version. If you do not want any of this information, pass NULL.

NWScanNCPExtensionInfo can return all of the information you need, eliminating the need to call NWSendNCPExtensionRequest. If all your information can be returned in the `queryData` parameter, obtain the buffer contents by calling NWGetNCPExtensionInfo (NLM). Receiving information this way does not call the NCP Extension handler and is useful only if a one-way server-to-client message is sufficient. If you do not need the information that is returned in the buffer, pass NULL.

See Also

NWDeRegisterNCPExtension (page 206), NWGetNCPExtensionInfo (NLM) (page 211), NWRegisterNCPExtension (page 226), NWSendNCPExtensionRequest (page 240)

NWSendNCPExtensionFraggedRequest

Sends a request to the specified NCP extension and allows data to be retrieved from and stored in noncontiguous memory locations

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM

Service: NCP Extension

Syntax

```
#include <nlm\nwncpx.h>

int NWSendNCPExtensionFraggedRequest  (
   LONG                          NCPExtensionID,
   const struct NCPExtensionMessageFrag
                                 *requestFrag,
   struct NCPExtensionMessageFrag
                                 * replyFrag);
```

Parameters

NCPExtensionID

(IN) Specifies the ID of the NCP Extension to process the request.

requestFrag

(IN) Points to the NCPExtensionMessageFrag structure containing information about the lengths and locations of the fragmented data for the NCP Extension handler to process (optional).

replyFrag

(IN/OUT) Points to the NCPExtensionMessageFrag structure. Inputs the maximum length of the data to return and where to place the data. Outputs the length of all returned data and where the data is stored (optional).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name and description
0	0x00	SUCCESSFUL The extension was found, and the non-null output parameters were filled.
126	0x7E	ERR_NCPEXT_TRANSPORT_PROTOCOL_VIOLATION The message transport mechanism entered a bad state in the protocol.

Value	Hex	Name and description
150	0x96	ERR_NO_ALLOC_SPACE There was not enough memory available on the server to allocate space for the message.
252	0xFC	ERR_NCPEXT_SERVICE_PROTOCOL_VIOLATION The service provider tried to return more data than the reply buffer could hold.
254	0xFE	ERR_NCPEXT_NO_HANDLER The NCP exception handler could not be found.
1-16		A communications error has occurred. (See niterror.h.)

Remarks

Call NWSendNCPExtensionFraggedRequest when you want to send your NCP Extension handler information stored at various locations which will avoid copying the information into a single buffer before sending it to your NCP Extension.

NWSendNCPExtensionFraggedRequest can also place the reply data into up to four specific locations, eliminating the need for you to copy the data from a reply buffer.

If your NCP Extension uses a single input buffer and/or a single output buffer, call NWSendNCPExtensionRequest instead of NWSendNCPExtensionFraggedRequest.

If your NLM registers its NCP Extension by a specific ID, use that ID when calling NWSendNCPExtensionFraggedRequest. If your NLM registers its NCP Extension by name, call NWGetNCPExtensionInfo (NLM) or NWScanNCPExtensions (NLM) to obtain the ID before calling NWSendNCPExtensionFraggedRequest.

NWSendNCPExtensionFraggedRequest copies the number of bytes from the server (indicated in the `totalMessageSize` field of the NCPExtensionMessageFrag structure), places them into memory locations (specified in the `fragList` field of the NCPExtensionMessageFrag structure), and sets a value to reflect the actual number of bytes transferred (indicated by the `totalMessageSize` field of the NCPExtensionMessageFrag structure).

NOTE: The information in the `replyFrag` parameter is valid only if NWSendNCPExtensionFraggedRequest returns SUCCESSFUL.

The request and reply buffers of the client must be reproduced on the server, so the maximum size of the buffers depends upon the memory available on the server that registers the NCP Extension. When NWSendNCPExtensionFraggedRequest is called, it attempts to allocate server memory for two message buffers. If it cannot allocate enough space, ERR_NO_ALLOC_SPACE will be returned. However, the request should be retried several times since server memory use is dynamic.

See Also

NWSendNCPExtensionRequest (page 240)

NWSendNCPExtensionRequest

Sends a request to the specified NCP extension

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM

Service: NCP Extension

Syntax

```
#include <nlm\nwncpx.h>

int NWSendNCPExtensionRequest  (
   LONG         NCPExtensionID,
   const void  *requestData,
   LONG         requestDataLen,
   void        *replyData,
   LONG        *replyDataLen);
```

Parameters

`NCPExtensionID`

(IN) Specifies the ID of the NCP Extension to process your request (obtained by calling the NWGetNCPExtensionInfo (NLM) or NWScanNCPExtensions (NLM) function.

`requestData`

(IN) Points to information for the NCP Extension handler to process (optional).

`requestDataLen`

(IN) Specifies the length (in bytes) of the input request buffer that is being sent to the NCP Extension (optional).

`replyData`

(OUT) Points to the information returned by the NCP Extension (optional).

`replyDataLen`

(IN/OUT) Points to the length (in bytes) of the `replyData` parameter. Outputs a pointer to the actual number of bytes placed into the buffer (optional).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name and description
0	0x00	SUCCESSFUL The extension was found, and the non-null output parameters were filled.
126	0x7E	ERR_NCPEXT_TRANSPORT_PROTOCOL_VIOLATION The message transport mechanism entered a bad state in the protocol.
150	0x96	ERR_NO_ALLOC_SPACE There was not enough memory available on the server to allocate space for the message.
252	0xFC	ERR_NCPEXT_SERVICE_PROTOCOL_VIOLATION The service provider tried to return more data than the reply buffer could hold.
254	0xFE	ERR_NCPEXT_NO_HANDLER The NCP exception handler could not be found.
1-16		A communications error has occurred. (See niterror.h.)

Remarks

NWSendNCPExtensionRequest sends the number of bytes specified by the `requestDataLen` parameter to the server.

If the `requestData` or `requestDataLen` parameter is set to NULL or zero respectively, no request data is sent.

NWSendNCPExtensionFraggedRequest copies the number of bytes from the server (indicated in the `totalMessageSize` field of the NCPExtensionMessageFrag structure), places them into memory locations (specified in the `fragList` field of the NCPExtensionMessageFrag structure), and sets a value to reflect the actual number of bytes transferred (indicated by the `totalMessageSize` field of the NCPExtensionMessageFrag structure).

NWSendNCPExtensionRequest copies the number of bytes from the server (specified in the `replyDataLen` parameter), places them into memory (specified in the `replyData` parameter), and sets a value to reflect the actual number of bytes transferred (specified by the `replyDataLen` parameter.

If the `replyData` or `replyDataLen` parameter is set to NULL or zero respectively, no reply data is returned.

NOTE: The information in the `replyData` parameter is valid only if NWSendNCPExtensionRequest returns SUCCESSFUL.

The request and reply buffers of the client must be reproduced on the server, so the maximum size of the buffers depends upon the memory available on the server that registers the NCP Extension. When NWSendNCPExtension is called, it attempts to allocate server memory for two message buffers. If it cannot allocate enough space, ERR_NO_ALLOC_SPACE will be returned. However, the request should be retried several times since server memory use is dynamic.

See Also

NWDeRegisterNCPExtension (page 206), NWGetNCPExtensionInfo (NLM) (page 211), NWScanNCPExtensions (NLM) (page 235), NWSendNCPExtensionFraggedRequest (page 238), NWRegisterNCPExtension (page 226)

NCP Extension Structures

14

This documentation alphabetically lists the NCP Extension structures and describes their purpose, syntax, and fields.

FragElement

Defines a fragment of a fragmented NCP extension request

Service: NCP Extension

Defined In: nwncpx.h

Structure

```
struct FragElement {
   void  *ptr ;
   LONG   size ;
};
```

Fields

ptr

Points to the fragment data.

size

Specifies the number of bytes that can be placed in the `ptr` field.

NCPExtensionClient

Defines an NCP extension client

Service: NCP Extension

Defined In: nwncpx.h

Structure

```
struct NCPExtensionClient {
   LONG   connection ;
   LONG   task ;
};
```

Fields

`connection`

Specifies the connection number of the client.

`task`

Specifies the task number of the client.

NCPExtensionMessageFrag

Defines a fragmented NCP extension request

Service: NCP Extension

Defined In: nwncpx.h

Structure

```
struct NCPExtensionMessageFrag {
   LONG                 totalMessageSize ;
   LONG                 fragCount ;
   struct FragElement    fragList [4];
};
```

Fields

`totalMessageSize`

Specifies the limit (in bytes) for the returned data.

`fragCount`

Specifies the number of FragElement structures stored in the `fragList` field.

`fragList`

Specifies an array of up to four FragElement structures.

Server-Based Connection Concepts

15

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

NOTE: Two service groups address connection issues: this one and Connection Number and Task Management Services. The functions in this group are mainly for obtaining connections to a server in the traditional way, by attaching or logging in and freeing them by logging out. The functions in the Connection Number and Task Management Services group are basically for switching between connections and task management. Both service groups have functions for getting information about connections and the objects currently using them.

An NLM application can work with connections in ways that are not available to workstation applications. By virtue of being loaded into the same server memory as the server, your NLM takes connection zero (0) and is granted automatic supervisory rights on the local server. Connection 0 gives your NLM unlimited access to the local servers 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 in order to gain access to a remote server. The Server-Based Connection functions allow an NLM to get a connection to a local or remote server.

15.1 Getting Connection Information

When an NLM calls LoginToFileServer, the server allocates the NLM a single connection by placing its object ID into one of the consecutively numbered slots in its connection table. That slot becomes the current thread groups connection number.

Many functions require you to specify a connection number, which you can get by calling either GetConnectionNumber or GetCurrentConnection.

15.2 LoginObject and LoginToFileServer

The difference between LoginToFileServer and LoginObject, a related function in the Connection Number and Task Management Services group, is that LoginObject requires a connection number and LoginToFileServer does not. So you would use LoginToFileServer when you want a new connection and LoginObject when you already have a connection.

15.3 Logout and LogoutFromFileServer

An NLM can log out from a server by calling any of the following functions:

- LogoutFromFileServer logs the NLM out of all connections to a *specified server* that the NLM is currently logged in to.
- Logout logs the NLM out of all connections on *all the servers* that the NLM is currently logged in to.

LogoutObject (Connection Number and Task Management Services) logs out the object logged in on the specified connection number on the current server (currently selected file server ID).

15.4 Unexpected Termination

Whenever a connection is terminated unexpectedly all server operations associated with that particular connection are cleaned up. Files and transactions are closed, locks are aborted, and network semaphores are freed. Hence, operations that were underway at the time of the unexpected termination must be restarted.

15.5 Getting Information

The functions in this group are for returning connection-related information, for example:

- the current connection number
- the current servers ID
- the specified servers ID
- all connection numbers in use on a specified server
- information about the object logged in on a particular connection, including its name and type
- an objects internet address
- the maximum number of connections
- the time an object logged in

15.6 Maximum Number of Connections Allowed

An NLM should always determine the maximum number of logical connections that are actually allowed on a server. Otherwise, it runs the risk of consuming too many connections, making it difficult for other objects on the network to obtain one.

The maximum number of connections allowed varies from server to server according to two factors: the servers version and the number of connections the particular site has purchased. For all practical purposes, NetWare 4.x has no limit on the number of connections available for use by NLM applications that are attached to the server. If on the other hand, the NLM logs into the server, the number of connections is restricted by the number purchased by the site. Unlike NetWare 4.x, NetWare 3.x provides 100 NLM connection numbers in all number-of-user versions of NetWare.

You can find out the maximum number of connections allowed by calling GetFileServerInformation (File Server Environment Services).

15.7 Server-Based Connection Functions

The descriptions of these functions use the terms station, connection, and connection number interchangeably:

Function	Description
AttachByAddress	Attaches an NLM (does not log it in) to the server specified by its addressing information
AttachToFileServer	Attaches an NLM (does not log it in) to the server specified by name

Function	Description
GetConnectionInformation	Returns information about the object logged in on the specified connection number
GetConnectionNumber	Returns the current connection number for the running process thread group
GetDefaultFileServerID	Returns the current file server ID for the running process thread group
GetFileServerID	Returns the file server ID of the specified server
GetInternetAddress	Returns a connections internet address (network number and node address)
GetLANAddress	Returns the 6-byte node address of a LAN board installed in a server
GetMaximumNumberOfStations	Returns the highest number of connections that have been allocated at any given time by a specified server
GetObjectConnectionNumbers	Returns a list of connection numbers that indicate how many times, and as what connection numbers, a Bindery object is logged in to the server
GetStationAddress	Returns the physical node address
GetUserNameFromNetAddress	Gets a user name (object ID) from an internet address
LoginToFileServer	Logs an object into the specified server and sets the current connection and current file server ID for the running processs thread group
Logout	Logs out all local and remote connections
LogoutFromFileServer	Logs out all connections to the specified server and sets the current connection and current file server ID to zero for the thread group that the running thread belongs to

An NLM can establish a connection to a server by logging in with LoginToFileServer, or by attaching with AttachToFileServer or AttachByAddress. These connections are terminated by calling Logout, LogoutFromFileServer, LogoutObject, LogoutFromFileServer or ReturnConnection (in Connection Number and Task Management Services).

Server-Based Connection Functions

16

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

AttachByAddress

Attaches (but does not log in) an NLM application to the server whose address is specified

Local Servers: blocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int AttachByAddress (
   int    transType,
   LONG   transLen,
   BYTE   *transBuf,
   WORD   *fileServerID);
```

Parameters

transType

(IN) Specifies the transport type, such as the IPX™ protocol.

transLen

(IN) Specifies the length of the transport buffer in bytes.

transBuf

(IN) Points to the transport buffer.

fileServerID

(OUT) Receives the file server ID of the newly attached server.

Return Values

If successful, this function returns zero. Otherwise, `NetWareErrno` is set.

Remarks

AttachByAddress attaches the NLM to the address described by the `transType`, `transLen`, and `transBuf` parameters. In addition AttachByAddress also sets the current connection of the calling thread (and of all threads in the same thread group) to be the new connection obtained by this function.

The file server ID of the attached server is returned in the `fileServerID` parameter.

The transport buffer contains different data depending on the transport type.

See Also

AttachToFileServer (page 254)

AttachToFileServer

Attaches (but does not log in) an NLM to the specified server

Local Servers: blocking

Remote Servers: blocking

Classification: 3.12, 3.2, 4.x, 5.x, 6.x

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int AttachToFileServer (
   char   *fileServerName,
   WORD   *fileServerID);
```

Parameters

`fileServerName`

(IN) Specifies a string containing the name of the server to which the NLM is to be attached.

`fileServerID`

(OUT) Receives the file server ID of the newly attached server.

Return Values

If successful, this function returns 0. Otherwise, `NetWareErrno` is set.

Remarks

AttachToFileServer attaches the NLM to the server specified by the `fileServerName` parameter. AttachToFileServer also sets the current connection of the calling thread (and of all threads in the same thread group) to be the new connection obtained by this function.

The `fileServerName` parameter is a string of no more than 48 characters, including the NULL terminator. The `fileServerID` parameter receives the file server ID of the attached server.

See Also

AttachByAddress (page 252)

GetConnectionID

See GetFileServerID (page 265)

GetConnectionInformation

Returns information about the object logged in as the specified connection number

Local Servers: nonblocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int GetConnectionInformation (
   WORD   connectionNumber,
   char   *objectName,
   WORD   *objectType,
   long   *objectID,
   BYTE   *loginTime);
```

Parameters

connectionNumber

(IN) Specifies the server connection number for which information is to be returned (0 to maximum connection number).

objectName

(OUT) Receives a string containing the name of the object logged in at the connection number (maximum 48 characters, including the NULL terminator).

objectType

(OUT) Receives the type of the object that is logged in at the connection number (OT_USER, OT_USER_GROUP, OT_PRINT_SERVER).

objectID

(OUT) Receives the unique ID of the object that is logged in at the connection number (0 = unused logical connection number).

loginTime

(OUT) Receives the date and time that the object logged in at the connection number (7 bytes).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name
0	(0x00)	ESUCCESS

Remarks

The `connectionNumber` is the position in the File Server Connection Table. A connection number of 0 is a special case. When 0 is specified, `objectName` returns the server name, `objectType` returns OT_FILE_SERVER, `objectID` is the supervisor's object ID, and `loginTime` is all zeros.

The `objectName` is an ASCIIZ string from 1 to 47 characters long.

The `objectType` identifies the object as an OT_USER (0x0001), OT_USER_GROUP (0x0002), OT_PRINT_SERVER (0x0007), and so on.

The `objectID` is the unique identification number of the object that is logged in at the logical connection number. (An object ID number of 0 means that an object is not logged in on the specified logical connection number.) This number uniquely identifies the object within the NetWare® Directory.

The `loginTime` is the date and time at which the object logged in. The 7 bytes contain the following information:

Byte	Contents	
0	Year	(0 to 99, where a value of 80 = 1980, 81 = 1981, etc.; however, if the value is less than 80, the year is considered to be in the twenty-first century.)
1	Month	(1 to 12)
2	Day	(1 to 31)
3	Hour	(0 to 23)
4	Minute	(0 to 59)
5	Second	(0 to 59)
6	Day	(0 to 6, where a value of 0 = Sunday, 1 = Monday, etc.)

See Also

GetConnectionList (page 260), GetConnectionNumber (page 262), GetInternetAddress (page 266), GetMaximumNumberOfStations (page 269), GetObjectConnectionNumbers (page 270), GetStationAddress (page 272)

GetConnectionInformation Example

```
#include <stdio.h>
#include <nwconn.h>

main()
{
   int    completionCode;
   WORD   objType, conn_no;
   char   objName[48];
   long   objID;
```

```
   BYTE   loginTime[7];

   printf ("Enter connection number:  ");
   scanf ("%u", &conn_no);
   completionCode = GetConnectionInformation (conn_no, objName,
&objType,
      &objID, loginTime);
   if (completionCode != 0)
      printf ("Error %d in GetConnectionInformation\n",
completionCode);
   else
   {
      printf ("Connection Number...        %d\n", conn_no);
      printf ("Object Name...              %s\n", objName);
      printf ("Object Type...              %u\n", objType);
      printf ("Object ID...                %ld\n", objID);
      printf ("Login Date...               %d/%d/%d\n",
      loginTime[1],
      loginTime[2],
      loginTime[0]);
      printf ("Login Time...               %d:%d:%d\n",
      loginTime[3],
      loginTime[4],
      loginTime[5]);
      switch (loginTime[6])
      {
      case 0:
      printf ("Login Day...          Sunday\n");
      break;

      case 1:
      printf ("Login Day...          Monday\n");
      break;

      case 2:
      printf ("Login Day...          Tuesday\n");
      break;

      case 3:
      printf ("Login Day...          Wednesday\n");
      break;

      case 4:
      printf ("Login Day...          Thursday\n");
      break;

      case 5:
      printf ("Login Day...          Friday\n");
      break;

      case 6:
      printf ("Login Day...          Saturday\n");
      break;
```

```
      default:
      printf ("Invalid day\n");
      break;
      }
   }
}
```

GetConnectionList

Returns a list of connections for a given object

Local Servers: blocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int GetConnectionList (
   LONG    objectID,
   LONG   lastConnection,
   LONG   *numberOfConnections,
   LONG   *connectionList,
   LONG   connectionSize);
```

Parameters

objectID

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

lastConnection

(IN) Specifies the connection number to start with (0 for the first call to this function).

numberOfConnections

(OUT) Receives the number of connections in `connectionList`.

connectionList

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

connectionSize

(IN) Specifies the maximum size allowed for `connectionList`.

Return Values

ESUCCESS or NetWare errors.

Remarks

This function returns an array of connection numbers (`connectionList`) for the object specified by `objectID`. The `numberOfConnections` parameter indicates the number of connection numbers in `connectionList`. The `connectionSize` parameter specifies the maximum size for `connectionList`.

Since the entire list of connections for an object might not be returned by one call, this function can be called iteratively to retrieve the entire list. For the first call, `lastConnection` should be 0. On subsequent calls, `lastConnection` should be given the last connection number in `connectionList`. When all connections have been retrieved, `connectionList` is not be completely filled.

See Also

GetConnectionNumber (page 262), GetConnectionInformation (page 256)

GetConnectionNumber

Returns the current connection number for the thread group that the running process belongs to

Local Servers: nonblocking

Remote Servers: nonblocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

WORD GetConnectionNumber (void);
```

Return Values

This function returns a connection number (0 to maximum connection number).

Remarks

This function is provided primarily for compatibility with the NetWare workstation APIs. In the NLM environment, this function returns the current connection number.

When the server allocates a connection number, it assigns the first unused connection number. When logging out, the server marks the connection number as unused but reserved in anticipation of reattachment. A reserved logical connection number is not reassigned until it becomes the first unused connection number and the attaching object has no connection number reserved.

Use GetFileServerInformation (File Server Environment Services) to find out how many logical connections a server can support.

See Also

GetConnectionList (page 260)

GetConnectionNumber Example

```
#include <nwconn.h>
WORD   connectionNumber;
connectionNumber = GetConnectionNumber ();
```

GetDefaultConnectionID

See GetDefaultFileServerID (page 264).

GetDefaultFileServerID

Returns the current file server ID for the thread group to which the running process belongs

Local Servers: nonblocking

Remote Servers: nonblocking

Classification: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int GetDefaultFileServerID (void);
```

Return Values

This function returns the current file server ID.

Remarks

This function returns the current file server ID as an int.

See Also

GetFileServerID (page 265)

GetDefaultFileServerID Example

```
#include <nwconn.h>
int   fileServerID;
fileServerID = GetDefaultFileServerID ( );
```

GetFileServerID

Returns the server ID of the specified server

Local Servers: nonblocking

Remote Servers: nonblocking

Classification: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int GetFileServerID (
   char   *fileServerName,
   WORD   *fileServerID);
```

Parameters

`fileServerName`

(IN) Specifies the name of the server (maximum 48 characters) for which `fileServerID` is returned.

`fileServerID`

(OUT) Receives the server number.

Return Values

The following table lists return values and descriptions.

Value	Hex	Name	Description
0		ESUCCESS	
252	(0xFC)	UNKNOWN_FILE_SERVER	Not attached to the specified server (it is not necessary to be logged in).

Remarks

The GetFileServerID function returns the file server ID of a server by passing the `fileServerName`, a NULL-terminated string.

If the NLM is not logged in to the named server, `fileServerID` receives a value of -1.

See Also

GetDefaultFileServerID (page 264)

GetInternetAddress

Returns a connection's Internet address

Local Servers: nonblocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int GetInternetAddress (
   WORD   connectionNumber,
   char   *networkNumber,
   char   *nodeAddress);
```

Parameters

connectionNumber

(IN) Specifies the connection number of the workstation or NLM for which an internetwork address is requested (1 to maximum number of connections).

networkNumber

(OUT) Receives the network number (4 bytes).

nodeAddress

(OUT) Receives the physical node address (6 bytes).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name	Description
0	(0x00)	ESUCCESS	
-1		EFAILURE	No network address is using that connection number.

Remarks

The internetwork address is comprised of the `networkNumber` and the `nodeAddress` and uniquely identifies an object. The `networkNumber` is the address of the network to which the object is attached. This is the same type of address as is displayed when the NetWare console command DISPLAY NETWORKS is issued.

The `nodeAddress` is the address of the object on the network that is returned in the `networkNumber` parameter. Use this address to send packets directly to the object with IPX/SPX™.

Print the `networkNumber` and the `nodeAddress` using the following format:

```
printf ("%08lx", LongSwap (* (long *) networkNumber) );
printf ("%08lx%04x", LongSwap (* (long *) nodeAddress),
   IntSwap (* (int *) nodeAddress + 4) );
```

See Also

GetConnectionInformation (page 256), GetConnectionNumber (page 262), GetMaximumNumberOfStations (page 269), GetObjectConnectionNumbers (page 270), GetStationAddress (page 272), IntSwap, LongSwap

GetInternetAddress Example

```
#include <nwconn.h>

int    completionCode;
char   networkNumber[4];
char   nodeAddress[6];

/* return the internet address of the current connection.*/
completionCode = GetInternetAddress (GetConnectionNumber(),
networkNumber,
   nodeAddress);
```

GetLANAddress

Returns the 6-byte node address of a LAN board installed in a server

Local Servers: nonblocking

Remote Servers: blocking

Classification: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int GetLANAddress (
   LONG    boardNumber,
   BYTE   *nodeAddress);
```

Parameters

boardNumber

(IN) Specifies the number of the LAN board for which the node address is desired. LAN board numbering begins with 1.

nodeAddress

(OUT) Receives the 6-byte node address of the selected LAN board.

Return Values

This function returns a value of 0 if successful. Otherwise, it returns a nonzero value.

Remarks

This function returns the 6-byte LAN address for the selected board for the current connection ID (current server).

See Also

GetInternetAddress (page 266), IpxGetInternetworkAddress (NLM)

GetMaximumNumberOfStations

(NetWare 4.x) returns the maximum number of connections that were allocated at any one time on the current server since it was last brought up; (NetWare 3.12 and earlier) returns the maximum number of connections licensed for that version of NetWare at that particular site

Local Servers: nonblocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int GetMaximumNumberOfStations (void);
```

Return Values

For NetWare 4.x, this function returns the maximum number of connections allocated *at one time* on the current server since it was last brought into service. Therefore, as more connections are made to the current server, the number returned by GetMaximumNumberOfStations increases. So if 10 connections have been allocated since the server was brought up, but 8 of those connections have logged out, making the number of current connections only 2, this function would return the number 10.

For NetWare 3.12 and earlier, this function returns the maximum number of connections allowed by the site license for the current server.

GetMaximumNumberOfStations Example

```
#include <nwconn.h>
int   maximumNumberOfStations;
maximumNumberOfStations = GetMaximumNumberOfStations ();
```

GetObjectConnectionNumbers

Returns a list of connection numbers that indicates how many times an object is logged in to the server and the connection number used for each login

Local Servers: nonblocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int GetObjectConnectionNumbers (
   char   *objectName,
   WORD   objectType,
   WORD   *numberOfConnections,
   WORD   *connectionList,
   WORD   maxConnections);
```

Parameters

objectName

(IN) Specifies the name of the object for which connection numbers are to be returned (maximum 48 characters, including the NULL terminator).

objectType

(IN) Specifies the type of the object for which connection numbers are returned (OT_USER, OT_USER_GROUP, OT_PRINT_SERVER).

numberOfConnections

(OUT) Receives the number of connections under which the object is logged in (0 to maximum number of connections). Zero indicates the given `objectName`/ `objectType` is not connected.

connectionList

(OUT) Receives a list of connection numbers under which the object is logged in.

maxConnections

(IN) Contains the maximum number of connection numbers to return in the `connectionList` parameter.

Return Values

The following table lists return values and descriptions.

Value	Hex	Name
0	(0x00)	ESUCCESS
NetWare Error		EFAILURE

Remarks

This function passes the `objectName`, `objectType`, and `maxConnections` parameters and returns the `numberOfConnections` and `connectionList` parameters.

When an object is logged in to a server from a workstation, NLM, or other program, the server places the object's ID number in a table. If an object logs in from three workstations and two NLM applications, its object ID number appears in the table five times.

The position of the workstation or NLM address in the table is a connection number. Each server can have a maximum number of connections. This function call returns a count and a list of the connection numbers under which the object is logged in.

If an NLM is logged in, its address is always the same as that of the server it is running on.

See Also

GetConnectionInformation (page 256), GetConnectionNumber (page 262), GetInternetAddress (page 266), GetMaximumNumberOfStations (page 269), GetStationAddress (page 272)

GetStationAddress

Returns the physical node address

Local Servers: nonblocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

void GetStationAddress (
   BYTE   *nodeAddress);
```

Parameters

nodeAddress

(OUT) Receives the node address (6 bytes).

Remarks

GetStationAddress always returns the node address as 0x000000000001. The reason for this is that the NetWare 3.x and 4.x OS uses an internal network number, and the server is node number 1 on the internal network.

In general, the `nodeAddress` uniquely identifies an object on a network. The internetwork address (network number and physical node address) uniquely identifies an object throughout an internetwork.

This function does not return the 4-byte `networkNumber`, which is also needed to send packets on the internet. The complete address of a station can be obtained by calling GetConnectionNumber and then calling GetInternetAddress.

See Also

GetConnectionNumber (page 262), GetInternetAddress (page 266)

GetStationAddress Example

```
#include <stdio.h>
#include <stdlib.h>
#include <nwconn.h>

main()
{
   int    ccode;
   BYTE   nodeAddress[6];
```

```
   GetStationAddress (nodeAddress);
   printf ("Station Address:      %02X%02X%02X%02X%02X%02X",
      nodeAddress[0],
      nodeAddress[1],
      nodeAddress[2],
      nodeAddress[3],
      nodeAddress[4],
      nodeAddress[5] );
}
```

GetUserNameFromNetAddress

Gets a user name (object ID) from an internet address

Local Servers: nonblocking

Remote Servers: N/A

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int GetUserNameFromNetAddress (
   BYTE                    * internetAddress,
   int                     sequenceNumber,
   struct UserNameStruct   * userNameP);
```

Parameters

internetAddress

(IN) Specifies a 10-byte internet address.

sequenceNumber

(IN) Specifies the number of users to skip.

userNameP

(OUT) Points to a structure of type UserNameStructure, which includes user name and object ID information.

Return Values

This function returns the next sequence number if successful. Otherwise, it returns a value of 0 if there are no more users with the specified address.

Remarks

The net address is a 10-byte address that specifies which users are to be returned. If all users are desired, the sequence number should initially be set to zero. The next sequence number to be used is then returned by the function until all users have been found. The returned `userNameP` parameter points to a structure of type UserNameStruct with the following fields:

```
BYTE    UserName[48]
LONG    ObjectID
```

See Also

GetInternetAddress (page 266), IpxGetInternetworkAddress (NLM)

LoginToFileServer

Logs an object in to the server

Local Servers: blocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int LoginToFileServer (
   char   *objectName,
   WORD   objectType,
   char   *objectPassword);
```

Parameters

objectName

(IN) Specifies the string containing the Bindery name of the object to be logged in (maximum 48 characters, including the NULL terminator).

A server name can be supplied as part of the `objectName`: `serverName`/ `objectName`

objectType

(IN) Specifies the Bindery type of the object to be logged in (OT_USER, OT_USER_GROUP, OT_PRINT_SERVER)

objectPassword

(IN) Specifies the string containing the object's password (maximum 128 characters, including the NULL terminator).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name
0	(0x00)	SUCCESSFUL
NetWare Error		EFAILURE

Remarks

If a server name is supplied as part of the `objectName`, an attempt is made to log in to the specified server. If successful, the new server is assigned a file server ID. The file server ID is made the current file server number, and the connection number on the new server is made the current

connection. Any logins using LoginToFileServer without specifying a new server name result in a login to the current server. If the login is not to the current server, the current working volume and current working path are set to the root.

To determine the connection number assigned to the NLM as a result of calling the LoginToFileServer function, an NLM can call GetCurrentConnection (defined in nwcntask.h) or GetConnectionNumber.

NOTE: LoginToFileServer establishes a single connection to the specified server. However, LogoutFromFileServer logs out all the connections to the specified server. You can use LogoutObject to log out a single connection.

See Also

LoginObject, Logout (page 277), LogoutFromFileServer (page 278), LogoutObject

LoginToFileServer Example

```
#include <stdio.h>
#include <stdlib.h>
#include <nwconn.h>

main()
{
   int   c;
   if(LoginToFileServer("supervisor",OT_USER,""))
   {
      printf("could not login\r\n");
      getch();
      return 1;
   }
   c = GetCurrentConnection();
   printf("current connection: %d\r\n",c);
   getch();
   LogoutObject(c);
   printf("logout performed\r\n");
   getch();
   ReturnConnection(c);            /* Only need to return connection
                                      if logged into local server */
   printf("connection returned\r\n");
}
```

Logout

Logs out all local and remote connections

Local Servers: blocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

void Logout (void);
```

Remarks

All connections on all servers are logged out. The current connection is set to 0 and the current working directory is set to the root of the SYS: volume.

NOTE: If you are using multiple threads, Logout logs out all other thread group and thread connections.

See Also

LoginToFileServer (page 275), LogoutFromFileServer (page 278), LogoutObject

LogoutFromFileServer

Logs out all connections on the specified server

Local Servers: blocking

Remote Servers: blocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

void LogoutFromFileServer (
   WORD   fileServerID);
```

Parameters

`fileServerID`

(IN) File server ID of the server whose connections are to be logged out.

Remarks

All connections on the specified file server ID (server number) are logged out. If logging out from the current server, the local server becomes the new current server. The current working volume and current working path are set to the root.

NOTE: If you are using multiple threads, Logout logs out all other thread group and thread connections.

See Also

LoginToFileServer (page 275), Logout (page 277), LogoutObject

NWGetSecurityLevel

Returns the current level of security the current NLM has for sending NCP packets

Local Servers: nonblocking

Remote Servers: nonblocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

 int NWGetSecurityLevel (
   void);
```

Return Values

NWGetSecurityLevel returns the current value the NLM has as its NCP packet signature option.

Remarks

The value returned by NWGetSecurityLevel corresponds with the values shown on the system console set command for NCPs, "NCP Packet Signature Option."

See Also

NWSetSecurityLevel (page 280)

NWSetSecurityLevel

Sets the current level of security the current NLM has for sending NCP packets

Local Servers: nonblocking

Remote Servers: nonblocking

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

Service: Server-Based Connection

Syntax

```
#include <nwconn.h>

int NWSetSecurityLevel (
   int   SecurityLevel);
```

Parameters

`SecurityLevel`

(IN) Specifies the new security level to be set.

Return Values

NWSetSecurityLevel returns the current NLMs most recent security level.

Remarks

The `SecurityLevel` parameter and the return value correspond with the values shown on the system console set command for NCPs, "NCP Packet Signature Option."

See Also

NWGetSecurityLevel (page 279)

Server-Based Message Concepts 17

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

Message functions enable applications to send broadcast messages (1 to 58 bytes) to specified target connections (workstations 1 to 250). The sending connection and the target connection must be attached to the same server. Broadcast messages use server processing time. For true peer-to-peer communication between programs across the network, applications can use Novell® IPX (Internetwork Packet Exchange) or SPX (Sequenced Packet Exchange) protocols, or NetBIOS. These protocols do not use server processing time and therefore promote better performance and greater flexibility (not limited to 55 bytes per message).

For broadcast messages, each connection on a server has a 58-byte message buffer associated with it.

Normally, when one connection sends a broadcast message to another connection, the server places the message in the target connections message buffer or pipe queue, and informs the target connection that a message has arrived.

17.1 Server-Based Message Functions

Function	Description
BroadcastToConsole	Broadcasts a message to the servers system console
DisableStationBroadcasts	Disables message reception
EnableStationBroadcasts	Enables message reception
GetBroadcastMessage	Returns a broadcast message from the current connection on the current server
SendBroadcastMessage	Sends a broadcast message to the specified connections on the current server

Server-Based Message Functions

18

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

BroadcastToConsole

Broadcasts a message to the server's system console (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWBroadcastToConsole (page 174))

Local Servers: blocking

Remote Servers: blocking

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

Service: Server-Based Message

Syntax

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

int BroadcastToConsole (
   char   *message);
```

Parameters

message

(IN) String containing the message to send (maximum 80 characters, including the NULL terminator).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name	Description
0	(0x00)	ESUCCESS	
252	(0xFC)	ERR_MESSAGE_QUEUE_FULL	
254	(0xFE)	ERR_IO_FAILURE	Lack of Dynamic Workspace

Remarks

The application must check that the message does not exceed 80 bytes and that it does not contain characters with ASCII values less than 0x20 or greater than 0x7E.

The server console displays a colon prompt followed by the message on a single line.

See Also

SendBroadcastMessage (page 288)

DisableStationBroadcasts

Disables message reception (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWDisableBroadcasts (page 176))

Local Servers: N/A

Remote Servers: blocking

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

Service: Server-Based Message

Syntax

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

int DisableStationBroadcasts (void);
```

Return Values

The following table lists return values and descriptions.

Value	Hex	Name
0	(0x00)	ESUCCESS
NetWare Error		UNSUCCESSFUL

Remarks

This function disables a connection from receiving broadcast messages from other connections.

See Also

EnableStationBroadcasts (page 286)

EnableStationBroadcasts

Enables message reception (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWEnableBroadcasts (page 178))

Local Servers: N/A

Remote Servers: blocking

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

Service: Server-Based Message

Syntax

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

int EnableStationBroadcasts (void);
```

Return Values

The following table lists return values and descriptions.

Value	Hex	Name
0	(0x00)	ESUCCESS
NetWare Error		UNSUCCESSFUL

Remarks

This function allows a connection or server console to receive broadcast messages from other connections.

See Also

DisableStationBroadcasts (page 285)

GetBroadcastMessage

Returns a broadcast message from the current connection on the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetBroadcastMessage (page 180))

Local Servers: N/A

Remote Servers: blocking

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

Service: Server-Based Message

Syntax

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

int GetBroadcastMessage (
   char   *messageBuffer);
```

Parameters

`messageBuffer`

(OUT) Returns a string containing the message (58 characters, including the NULL terminator).

Return Values

The following table lists return values and descriptions.

Value	Hex	Name	Description
0	(0x00)	ESUCCESS	
253	(0xFC)	ERR_MESSAGE_QUEUE_FULL	
254	(0xFE)	ERR_IO_FAILURE	Lack of Dynamic Workspace

Remarks

If no message is pending, `messageBuffer[0]` contains a NULL. Only one message can be stored. GetBroadcastMessage cannot be called iteratively to poll messages.

Only two broadcast modes are possible: enabled and disabled.

See Also

SendBroadcastMessage (page 288)

SendBroadcastMessage

Sends a broadcast message to the specified logical connections on the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (*NDK: NLM Development Concepts, Tools, and Functions*) and call NWSendBroadcastMessage (page 182))

Local Servers: N/A

Remote Servers: blocking

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

Service: Server-Based Message

Syntax

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

int SendBroadcastMessage (
   char   *message,
   WORD   *connectionList,
   BYTE   *resultList,
   WORD   connectionCount);
```

Parameters

`message`

(IN) String containing the message to be sent (58 characters, including the NULL terminator).

`connectionList`

(IN) List of connection numbers to which the message is to be broadcast.

`resultList`

(OUT) Receives a result (status) code for each connection number contained in `connectionList`.

`connectionCount`

(IN) Maximum number of connections to which the requesting workstation wants to send the message.

Return Values

The following table lists return values and descriptions.

Value	Hex	Name	Description
0	(0x00)	ESUCCESS	
150	(0x96)	ERR_SERVER_OUT_OF_MEMORY	
254	(0xFE)	ERR_IO_FAILURE	Lack of Dynamic Workspace

Remarks

The application must check that the message does not exceed 58 characters and that it does not contain characters with ASCII values less than 0x20 or greater than 0x7E.

The `connectionList` parameter specifies the connection number of each station. Each byte in the `connectionList` field has a corresponding byte in `resultList`.

The `resultList` parameter returns a result code for each connection number contained in the `connectionList` field. The following result codes are defined:

Code	Description
0x00	Successful. The server stored the message in the target connection's message buffer. (It is the target connection's responsibility to retrieve and display the message.)
0xFC	Rejected. The target connection's message buffer is already holding a message.
0xFD	Invalid Connection Number. The specified connection number is unknown.
0xFF	Blocked. The target connection's message mode is set to block messages, or the target connection is not in use.

See Also

GetBroadcastMessage (page 287), GetConnectionInformation (page 256), GetConnectionNumber (page 262)

Revision History

A

The following table outlines all the changes that have been made to the Connection, Message, and NCP Extensions documentation (in reverse chronological order):

Release Date	Revision Description
March 1, 2006	Added a link from GetCurrentConnection (page 157) to information about connection numbers.
October 5, 2005	Transitioned to revised Novell documentation standards.
March 2, 2005	Fixed legal information.
October 6, 2004	Added new error code values to the NWCCOpenConnByName (page 55) function.
June 9, 2004	Updated the front material.
June 2003	Updated the Remarks section for the NWRegisterNCPExtension (page 226) and NWRegisterNCPExtensionByID (page 230) functions. Changed all Pascal references to Delphi references. Added a new field to the NWCCConnInfo (page 125) structure and a infoType flag to retrieve the identityHandle.
October 2002	Updated the Pascal syntax for the structures.
May 2002	Updated the example of using an IPX Internal Network Number in NWCCTranAddr (page 129).
	Removed the IPX only statements from AttachByAddress (page 252).
	Fixed minor punctuation issues.
February 2002	Updated links and Pascal syntaxes.
September 2001	Added support for NetWare 6.0 to documentation.
June 2001	Made changes to improve document accessibility.
	Added links to table in Section 9.3, “Message Functions,” on page 172.
	Added links to table in Section 11.13, “NCP Extension Functions,” on page 196.
May 2001	Replaced references to ncpext.h with nwncpx.h in FragElement (page 244), NCPExtensionClient (page 245), and NCPExtensionMessageFrag (page 246).
	Added NWCC_NAME_FORMAT_NDS to Section 5.9, “Name Format Values,” on page 138.
	Added several values to Section 5.2, “Connection State Values,” on page 135.

Release Date	Revision Description
February 2001	Changed NWGetNCPExtensionInfo (NLM) (page 211) so that the second parameter is a LONG * instead of a LONG, as per include\nlm\nwncpx.h. Changed the Pascal syntaxes of NWCCOpenConnByAddr (page 53), NWCCOpenConnByPref (page 58), and NWCCOpenConnByRef (page 60). Went through the syntaxes of all NCP Extension functions and changed any inaccurate include statements (especially among NLM only functions). Changed "bindery object" to "object" references since these references can also apply to NDS objects.
July 2000	Added information about IP address formats in NWCCTranAddr (page 129) structure. Noted that NWCC_INFO_RETURN_ALL is not a valid value for the infoType parameter of the NWCCGetConnInfo (page 35) function.
May 2000	Added this revision history