# Novell Developer Kit

www.novell.com

SERVER MANAGEMENT

March 1, 2006

Novell®

## Legal Notices

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

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

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



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

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

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

## Novell Trademarks

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

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

TTS is a trademark of Novell, Inc.

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

Virtual Loadable Module is a trademark of Novell, Inc.

VLM is a trademark of Novell, Inc.

Yes Certified is a trademark of Novell, Inc.

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

## Third-Party Materials

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

# Contents

# About This Guide

This guide contains information about concepts, tasks, functions, and structures having to do with server management. Topics discussed include server environment, server management, and TTS. Volume management is discussed separately in the *NDK: Volume Management* documentation.

This guide documents the following services:



### Feedback

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

### Documentation Updates

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

### Additional Information

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

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

For CLib and XPlat 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 Forum (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.

# Server Environment Concepts

1

This documentation provides and overview of Server Environment, its functions, and features.

## 1.1 Server Environment Introduction

Server Environment returns detailed statistical information about NetWare® servers. It also allows you to perform the following tasks:

- Check the version of NetWare running on the server
- Check whether you have console operator privileges on a server
- Bring down a server
- Enable and disable server logins
- Attach, log in, and log out of NetWare servers using the bindery

The functions fall into three groups:

- Functions compatible with 3.x and above
- Functions compatible with 4.x, 5.x, and 6.x

The header nwfse.h defines NetWare 4.x, 5.x, and 6.x functions; nwserver.h defines the remaining functions.

For a description of structures and other data definitions that relate to this topic, see “Server Environment Structures” on page 171.

## 1.2 Server Get Functions for 4.x-6.x

These functions return 4.x, 5.x, and 6.x server information by server name and type.

| Function | Header | Comment |
|---|---|---|
| NWGetServerInfo | nwfse.h | Returns information about the server including the specified type and name. |
| NWGetServerSourcesInfo | nwfse.h | Returns information about all servers matching the specified type and name. |
| NWGetKnownServersInfo | nwfse.h | Returns all known servers. |

## 1.3 Server Information Functions for 4.x-6.x

These functions return detailed information about the NetWare® server associated with the specified connection handle.

| Function | Header | Comment |
|---|---|---|
| NWGetCacheInfo | nwfse.h | Returns information about a server's file cache. |
| NWGetFileServerInfo | nwfse.h | Returns server operation statistics information. |
| NWGetNetWareFileSystemsInfo | nwfse.h | Returns counters of the times specific operations on the file system were performed. |
| NWGetPacketBurstInfo | nwfse.h | Returns counters and statistics about packet burst on the server. |
| NWGetIPXSPXInfo | nwfse.h | Returns a server's internal IPX and SPX statistics. |
| NWGetGarbageCollectionInfo | nwfse.h | Returns counters about a server's memory allocation manager. |
| NWGetDirCacheInfo | nwfse.h | Returns statistics about a server's directory caching. |
| NWGetCPUInfo | nwfse.h | Returns CPU information and descriptive strings for the CPU type, numeric coprocessor, and bus type for the indicated CPU number (for 4.x, this is 1). |
| NWGetVolumeSwitchInfo | nwfse.h | Returns counters containing the number of times a specified code path was taken in the server. |
| NWGetOSVersionInfo | nwfse.h | Returns version information about the server's operating system. |

# 1.4 Server LAN Board Information Functions for 4.x-6.x

These functions return information about LAN boards on a 4.x, 5.x, or 6.x server.

| Function | Header | Comment |
|---|---|---|
| NWGetActiveLANBoardList | nwfse.h | Returns a list of LAN Board IDs that can be used as input to other functions. |
| NWGetLANConfigInfo | nwfse.h | Returns configuration information for a LAN Board. |
| NWGetLANCommonCountersInfo | nwfse.h | Returns counters common to all types of LAN Boards. |
| NWGetLANCustomCountersInfo | nwfse.h | Returns counters specific to a particular LAN Board. |
| NWGetLSLInfo | nwfse.h | Returns information about the link support layer (LSL). |

| Function | Header | Comment |
|---|---|---|
| NWGetLSLLogicalBoardStats | nwfse.h | Returns information about the LSL logical boards. |

## 1.5 Server Media Manager Information Functions for 4.x-6.x

These functions return media manager information for a 4.x, 5.x, or 6.x server.

| Function | Header | Comment |
|---|---|---|
| NWGetLoadedMediaNumList | nwfse.h | Returns a list of Media IDs for all the managed media objects in a server. |
| NWGetMediaMgrObjList | nwfse.h | Returns a list of Media IDs for all the media objects matching the specified type. |
| NWGetMediaMgrObjChildrenList | nwfse.h | Returns a list of children IDs for a media object. |
| NWGetMediaMgrObjInfo | nwfse.h | Returns information about a media object. This information includes parent, sibling, and children counts and I/O capabilities. |
| NWGetMediaNameByMediaNum | nwfse.h | Returns the descriptive name and information about a media object specified by the media ID. |

## 1.6 Server Network and Router Information Functions for 4.x-6.x

These functions return routing and service advertising information for a 4.x, 5.x, or 6.x server.

| Function | Header | Comment |
|---|---|---|
| NWGetKnownNetworksInfo | nwfse.h | Returns a list of networks known to a server. |
| NWGetNetworkRouterInfo | nwfse.h | Returns information about the specified network to a server, if known. |
| NWGetGeneralRouterAndSAPInfo | nwfse.h | Returns flags and information concerning the status of routing and SAP on a server. |
| NWGetNetworkRoutersInfo | nwfse.h | Returns information about routers on the specified network. |

## 1.7 Server NLM Information Functions for 4.x-6.x

These functions return information about NLMs on a 4.x, 5.x, or 6.x server.

| Function | Header | Comment |
|---|---|---|
| NWGetNLMLoadedList | nwfse.h | Returns a list of NLM IDs that can be used with NWGetNLMInfo and NWGetNLMsResourceTagList. |
| NWGetNLMInfo | nwfse.h | Returns strings identifying an NLM's filename, name, and copyright, along with detailed information about the module. |
| NWGetNLMsResourceTagList | nwfse.h | Returns resource tag lists for an NLM. Resource tags are used by NetWare® to identify resources allocated by the module. |

# 1.8 Server Protocol Stack Information Functions for 4.x-6.x

These functions return protocol stack information for a 4.x, 5.x, or 6.x server.

| Function | Header | Comment |
|---|---|---|
| NWGetActiveProtocolStacks | nwfse.h | Returns a list of Stack IDs for all the loaded protocol stacks in the server. |
| NWGetProtocolStackConfigInfo | nwfse.h | Returns configuration information describing a protocol stack. |
| NWGetProtocolStackStatsInfo | nwfse.h | Returns counters for a protocol stack, including the number of custom counters. |
| NWGetProtocolStkNumsByMediaNum | nwfse.h | Returns Stack IDs for the specified media number. |
| NWGetProtocolStkNumsByLANBrdNum | nwfse.h | Returns Stack IDs for the protocol stacks bound to a LAN Board. |
| NWGetProtocolStackCustomInfo | nwfse.h | Returns the custom counters for the protocol stack. |

# 1.9 Server Set Functions for 4.x-6.x

These functions return the set table configuration categories and commands for a 4.x, 5.x, or 6.x server.

| Function | Header | Comment |
|---|---|---|
| NWGetServerSetCommandsInfo | nwfse.h | Returns all of a server's set table commands for all categories. |
| NWGetServerSetCategories | nwfse.h | Returns the set table categories on the server. |

## 1.10 Server Volume Information Functions for 4.x-6.x

These functions return volume information for a 4.x, 5.x, or 6.x server.

| Function | Header | Comment |
|---|---|---|
| NWGetVolumeSegmentList | nwfse.h | Returns a list of volume segment information for the server. |
| NWGetVolumeInfoByLevel | nwfse.h | Returns detailed information about the specified volume according to the specified information level. |

## 1.11 Server Configuration Functions

These functions read NetWare® server configuration data.

| Function | Header | Comment |
|---|---|---|
| NWCheckNetWareVersion | nwserver.h | Allows verification of compatibility between applications and the version of NetWare running on a NetWare server. |
| NWGetFileServerDateAndTime | nwserver.h | Returns the network date and time maintained on the specified NetWare server. |
| NWGetFileServerDescription | nwserver.h | Returns descriptive information about the specified NetWare server, including company name, version, revision date, and copyright notice. |
| NWGetFileServerExtendedInfo | nwserver.h | Returns extended information about the specified NetWare server. |
| NWGetFileServerInformation | nwserver.h | Returns commonly referenced information about a NetWare server including version numbers, connection statistics, SFT level, and TTS level. |
| NWGetFileServerLoginStatus | nwserver.h | Returns whether clients can log in to the specified workstation. |
| NWCCGetConnRefInfo | nwserver.h | Returns the name of the NetWare server associated with the specified connection ID. |
| NWGetFileServerVersionInfo | nwserver.h | Returns name and version information for the specified NetWare server. |
| NWGetNetworkSerialNumber | nwserver.h | Returns the NetWare server's serial number and the application number. |
| NWIsManager | nwserver.h | Checks whether a calling station is a manager |

## 1.12 Server Connection Functions

These functions perform NetWare® server attachments and logins.

| Function | Header | Comment |
|---|---|---|
| NWAttachToFileServer | nwserver.h | Attempts to establish a connection with the specified NetWare server. |
| NWAttachToFileServerByConn | nwserver.h | Attaches to a NetWare server through a service identified by a connection. |
| NWLoginToFileServer | nwserver.h | Attempts to log a bindery object in to a NetWare server. This function performs only the login; the workstation must be currently attached to the server. |
| NWLogoutFromFileServer | nwserver.h | Attempts to log a bindery object out of the specified NetWare server. This function doesn't release the connection. |

## 1.13 Server Console Functions

These functions perform console operations.

| Function | Header | Comment |
|---|---|---|
| NWCheckConsolePrivileges | nwserver.h | Determines whether the client is a NetWare® server console operator. |
| NWDisableFileServerLogin | nwserver.h | Disables all logins to a NetWare server. |
| NWDownFileServer | nwserver.h | Brings a NetWare server down. |
| NWEnableFileServerLogin | nwserver.h | Enables logins to a NetWare server. |
| NWSetFileServerDateAndTime | nwserver.h | Sets the date and time of a NetWare server. |

# Server Environment Functions

# 2

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

# GetServerConfigurationInfo

Returns the engine type and loader type of the server.

**Local Servers:** nonblocking

**Remote Servers:** N/A

**NetWare Server:** 3.12, 3.2, 4.02, 4.1, 5.x, 6.x

**Platform:** NLM

**Service:** Server Environment

## Syntax

```
#include <nwenvrn.h>

int GetServerConfigurationInfo  (
   int  *serverType,
   int  *loaderType);
```

## Parameters

**`serverType`**

(OUT) Points to the NetWare server engine type.

**`loaderType`**

(OUT) Points to the NetWare loader type.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | 0x00 | ESUCCESS always. |

## Remarks

You can pass a NULL pointer in either parameter. When a NULL pointer is passed in, a value will not be returned for the specified parameter.

`serverType` receives one of the following values defined in NWENVRN.H:

| Value | Constant | Description |
|---|---|---|
| 0 | TYPE_NORMAL_SERVER | NLM application is running on a normal NetWare server. |
| 1 | TYPE_IO_ENGINE | NLM application is running in the IO Engine of a NetWare SFT III™ server. |

| Value | Constant | Description |
|---|---|---|
| 2 | TYPE_OS_ENGINE | NLM application is running in the MS Engine of a NetWare SFT III server. |

`loaderType` receives one of the following values defined in NWENVRN.H:

| Value | Constant | Description |
|---|---|---|
| 1 | LOADER_TYPE_DOS | NLM application is running on a dedicated NetWare server with a DOS loader. |
| 2 | LOADER_TYPE_OS2 | NLM application is running on a nondedicated server running on OS/2. |
| 3 | LOADER_TYPE_MSWIN31 | NLM application is running on a nondedicated server running Netware for Windows. |

## Example

### GetServerConfigurationInfo

```
#include <stdio.h>
#include <nwenvrn.h>

void main()
{
   int  serverType, loaderType;

   if (!GetServerConfigurationInfo(&serverType, &loaderType))
   {
       if (loaderType == LOADER_TYPE_OS2)
          printf("This NLM is running on NetWare for OS/2.\n\n");
       else if (loaderType == LOADER_TYPE_DOS)
       {
          if (serverType == TYPE_IO_ENGINE)
             printf("This NLM is running on NetWare SFTIII"
                    " in an IO Engine.\n\n");
          else if (serverType == TYPE_OS_ENGINE)
             printf("This NLM is running on NetWare SFTIII"
                    " in an MS Engine.\n\n");
          else if (serverType == TYPE_NORMAL_SERVER)
             printf("This NLM is running on a dedicated"
                    " NetWare server with a DOS loader.\n\n");
       }
    }
}
```

# NWAttachToFileServer

Attaches to 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:** Server Environment

## Syntax

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

NWCCODE N_API NWAttachToFileServer (
   const nstr8 N_FAR    *serverName,
   nuint16               scopeFlag,
   NWCONN_HANDLE N_FAR  *newConnID);
```

## Delphi Syntax

```
uses calwin32

Function NWAttachToFileServer
  (const serverName : pnstr8;
   scopeFlag : nuint16;
   Var newConnID : NWCONN_HANDLE
) : NWCCODE;
```

## Parameters

**serverName**

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

**scopeFlag**

Is reserved; must be 0.

**newConnID**

(OUT) Points to the new connection handle, if the attachment was successful.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8800 | ALREADY_ATTACHED |
| 0x8801 | INVALID_CONNECTION |
| 0x8847 | NO_SERVER_ERROR |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89FC | UNKNOWN_FILE_SERVER |
| 0x89FF | NO_RESPONSE_FROM_SERVER |

## Remarks

**WARNING:** Before attaching to the specified server, NWAttachToFileServer tries to get the server's net address from the default server's Bindery.

NO_RESPONSE_FROM_SERVER will be returned if RIP traffic is filtered on a router but SAP traffic is not. The dynamic object can be read from the bindery, but the request to attach to a server cannot be routed

## NCP Calls

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

# NWAttachToFileServerByConn

Attaches to a NetWare server through a service identified by a 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:** Server Environment

## Syntax

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

NWCCODE N_API NWAttachToFileServerByConn (
   NWCONN_HANDLE         conn,
   const nstr8 N_FAR    *serverName,
   nuint16               scopeFlag,
   NWCONN_HANDLE N_FAR  *newConnID);
```

## Delphi Syntax

```
uses calwin32

Function NWAttachToFileServerByConn
  (conn : NWCONN_HANDLE;
   const serverName : pnstr8;
   scopeFlag : nuint16;
   Var newConnID : NWCONN_HANDLE
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle through which to attach.

**serverName**

(IN) Points to a 48-character buffer for the server name (optional).

**scopeFlag**

(IN) Reserved for Novell use only; must be 0.

**newConnID**

(OUT) Points to the connection handle, if any, to `serverName`.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8800 | ALREADY_ATTACHED |
| 0x8801 | INVALID_CONNECTION |
| 0x8847 | NO_SERVER_ERROR |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89FC | UNKNOWN_FILE_SERVER |

## Remarks

NWAttachToFileServerByConn allows attachments to servers not seen by the preferred server.

## NCP Calls

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

## See Also

NWAttachToFileServer (page 28)

# NWCheckConsolePrivileges

Determines if the logged-in user is a console operator.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWCheckConsolePrivileges  (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

Function NWCheckConsolePrivileges
  (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 |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

## NCP Calls

0x2222 23 200 Check Console Privileges

# NWCheckNetWareVersion

Checks compatibility of OS modules.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWCheckNetWareVersion
  (NWCONN_HANDLE   conn,
   nuint16         minVer,
   nuint16         minSubVer,
   nuint16         minRev,
   nuint16         minSFT,
   nuint16         minTTS,
   pnuint8         compatibilityFlag);
```

## Delphi Syntax

```
uses calwin32

Function NWCheckNetWareVersion
  (conn : NWCONN_HANDLE;
   minVer : nuint16;
   minSubVer : nuint16;
   minRev : nuint16;
   minSFT : nuint16;
   minTTS : nuint16;
   compatibilityFlag : pnuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the connection handle of the server to check.

**minVer**

(IN) Specifies the minimum version required for the module to run.

**minSubVer**

(IN) Specifies the minimum sub-version required for the module to run.

**minRev**

(IN) Specifies the minimum revision required for the module to run.

**minSFT**

(IN) Specifies the minimum revision required to check System Fault Tolerance (SFT).

**minTTS**

(IN) Specifies the minimum revision required to check Transaction Tracking System (TTS).

**compatibilityFlag**

(OUT) Points to a flag indicating compatibility:

| C Value | Delphi Value | Value Name |
|---|---|---|
| 0x00 | $00 | COMPATIBLE |
| 0x01 | $01 | VERSION_NUMBER_TOO_LOW |
| 0x02 | $02 | SFT_LEVEL_TOO_LOW |
| 0x04 | $04 | TTS_LEVEL_TOO_LOW |

## Return Values

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

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

## NCP Calls

0x2222 23 200 Check Console Privileges

## See Also

NWGetFileServerVersionInfo (page 74)

# NWDisableFileServerLogin

Allows an operator to instruct the NetWare server to refuse new login requests.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWDisableFileServerLogin  (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

Function NWDisableFileServerLogin
  (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 |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

## Remarks

NWDisableFileServerLogin is usually made during some crucial time-before taking the server down, for instance.

It is recommended that caution be used with NWDisableFileServerLogin. If, after calling NWDisableFileServerLogin, the service connection to the NetWare server is lost or destroyed, a new connection cannot be created; therefore, the user cannot log in again. If no other user on the server has SUPERVISOR privileges, the server must be brought down from the console connected to the server and rebooted before any new users (including the SUPERVISOR) can access it.

To call NWDisableFileServerLogin, you must have console operator rights.

## NCP Calls

0x2222 23 203 Disable File Server Login

# NWDownFileServer

Allows a supervisor to bring down a NetWare server from a remote 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWDownFileServer  (
   NWCONN_HANDLE   conn,
   nuint8          forceFlag);
```

## Delphi Syntax

```
uses calwin32

Function NWDownFileServer
  (conn : NWCONN_HANDLE;
   forceFlag : nuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**forceFlag**

(IN) Specifies a flag enabling the server to shut down when files are still open (0=enabled).

## 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 |
| 0x89FF | Down Failure |

## Remarks

If `forceFlag` is zero, the server shuts down even if files are open. If `forceFlag` is non-zero and any files are in use or open, 0x89FF (failure) is returned, and the server stays up. If no files are open or in use, the server shuts down, and SUCCESSFUL returns.

To call NWDownFileServer, you must have console operator rights.

## NCP Calls

0x2222 23 211 Down File Server

# NWEnableFileServerLogin

Allows an operator to instruct the server to begin accepting new login requests from clients.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWEnableFileServerLogin  (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

Function NWEnableFileServerLogin
  (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 |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

## Remarks

Enabling the server's log state also unlocks the SUPERVISOR's account if it has been locked because of intruder detection.

To call NWEnableFileServerLogin, you must have console operator rights.

If the calling station does not have operator privileges, NO_CONSOLE_PRIVILEGES is returned, and the NetWare server's log state remains unchanged.

## NCP Calls

0x2222 23 204 Enable File Server Login

## See Also

NWDisableFileServerLogin (page 35)

# NWEnumNetAddresses

Enumerates all the network addresses used by the specified NetWare server.

**Local Servers:** blocking

**Remote Servers:** blocking

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

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

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

**Service:** Server Environment

## Syntax

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

NWCCODE NWEnumNetAddresses (
   NWCONN_HANDLE                        conn,
   pnuint32                             searchNumber
   SERVER_AND_VCONSOLE_INFO N_FAR      *serverTimeAndVConsoleInfo,
   pnuint16                             reserved,
   NW_GUID N_FAR                       *fseServerGUID,
   nuint32                              itemsInArray,
   pnuint32                             itemsReturned,
   NWFSE_NETWORK_ADDRESS N_FAR         *fseNetworkAddresses);
```

## Delphi Syntax

```
uses calwin32

Type
   NW_GUID = Array[0..15] of nuint8;

Function NWEnumNetAddresses(
  conn : NWCONN_HANDLE;
  Var searchNumber : nuint32;
      Var serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
      reserved : pnuint16;
      Var fseServerGUID : NW_GUID;
      itemsInArray : nuint32;
  Var itemsReturned : nuint32;
  Var fseNetworkAddresses: NWFSE_NETWORK_ADDRESS
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**searchNumber**

(IN/OUT) Points to the iteration value used for subsequent calls (set to zero initially).

**serverTimeAndVConsoleInfo**

(OUT) Points to SERVER_AND_VCONSOLE_INFO, which contains the server console version.

**reserved**

Is reserved for future use.

**fseServerGUID**

(OUT) Points to NW_GUID, which contains the server's Global Universal Identification (GUID).

**itemsInArray**

(IN) Specifies the size of the array pointed to by `fseNetworkAddresses`.

**itemsReturned**

(OUT) Points to the actual number of addresses returned by `fseNetworkAddresses`.

**fseNetworkAddresses**

(OUT) Points to NWFSE_NETWORK_ADDRESS, which contains the network address information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x880E | NWE_BUFFER_OVERFLOW |
| 0x8977 | NWE_BUFFER_TOO_SMALL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

You need to allocate enough space for the address in the `address` field of NWFSE_NETWORK_ADDRESS, or NWE_BUFFER_OVERFLOW will be returned.

Upon return, the `addressSize` field in NWFSE_NETWORK_ADDRESS will contain the size needed to read the address. To call NWEnumNetAddresses iteratively and avoid the NWE_BUFFER_OVERFLOW error, reset the `addressSize` field to the maximum buffer size.

## NCP Calls

0x2222 123 17 Enumerate NCP Service Network Addresses

# NWGenerateGUIDs

Returns a Global Universal Identification (GUID) list from a server.

**Local Servers:** blocking

**Remote Servers:** blocking

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

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

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

**Service:** Server Environment

## Syntax

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

NWCCODE NWGenerateGUIDs (
   NWCONN_HANDLE   connHandle,
   nuint32         GUIDSize,
   NW_GUID         GUIDList[]);
```

## Delphi Syntax

```
uses calwin32

Function NWGenerateGUIDs
  (connHandle : NWCONN_HANDLE;
   GUIDSize : nuint32;
   GUIDList [] :  NW_GUID
) : NWCCODE;
```

## Parameters

**`connHandle`**

(IN) Specifies the connection handle of the server where the GUID list request should be sent.

**`GUIDSize`**

(IN) Specifies the number of GUIDs that will fit in the space allocated for `GUIDList`.

**`GUIDList`**

(OUT) Points to an array of NW_GUID structures that contain a list of the server's GUIDs.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x880E | NWE_BUFFER_OVERFLOW |
| 0x8869 | ACCESS_VIOLATION |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89FB | INVALID_PARAMETER |

## Remarks

`GUIDSize` will be the maximum number of GUIDs generated by the server.

You need to allocate enough space to accommodate the number of GUIDs specified in `GUIDSize`.

## NCP Calls

0x2222 23 33 Generate GUIDs

# NWGetActiveConnListByType

Returns a bitmap (set if logged in) of all connections of a specified type.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetActiveConnListByType  (
   NWCONN_HANDLE                    conn,
   nuint32                          startConnNum,
   nuint32                          connType,
   NWFSE_ACTIVE_CONN_LIST  N_FAR   *fseActiveConnListByType);
```

## Delphi Syntax

```
uses calwin32

Function NWGetActiveConnListByType
  (conn : NWCONN_HANDLE;
   startConnNum : nuint32;
   connType : nuint32;
   Var fseActiveConnListByType : NWFSE_ACTIVE_CONN_LIST
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startConnNum**

(IN) Specifies the first connection number to return information about.

**connType**

(IN) Specifies the type of the connection (see Connection Type Values (*NDK: Connection, Message, and NCP Extensions*)).

**fseActiveConnListByType**

(OUT) Points to NWFSE_ACTIVE_CONN_LIST.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89FD | Invalid Connection |
| 0x89FF | Failure or Invalid Start Number |

## Remarks

Console operator rights are NOT necessary to call NWGetActiveConnListByType.

## NCP Calls

0x2222 123 14 Get Active Connection List By Type

## See Also

NWGetUserInfo (page 155)

# NWGetActiveLANBoardList

Returns information about the active LAN boards on a 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetActiveLANBoardList  (
   NWCONN_HANDLE                         conn,
   nuint32                               startNum,
   NWFSE_ACTIVE_LAN_BOARD_LIST  N_FAR   *fseActiveLANBoardList);
```

## Delphi Syntax

```
uses calwin32

Function NWGetActiveLANBoardList
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   Var fseActiveLANBoardList : NWFSE_ACTIVE_LAN_BOARD_LIST
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the starting LAN board number.

**fseActiveLANBoardList**

(OUT) Points to NWFSE_ACTIVE_LAN_BOARD_LIST.

## 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 |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89FF | Failure or Invalid Start Number |

## Remarks

Console operator rights are NOT necessary to call NWGetActiveLANBoardList.

`startNum` will normally be 0, unless the amount of LAN Boards is greater than FSE_MAX_NUM_OF_LANS. To return the extra, set `startNum` to LANLoadedCount + 1.

## NCP Calls

0x2222 123 20 Active LAN Board List

## See Also

NWGetLANCommonCountersInfo (page 89), NWGetLANConfigInfo (page 91), NWGetLANCustomCountersInfo (page 93), NWGetLSLLogicalBoardStats (page 99), NWGetProtocolStkNumsByLANBrdNum (page 139)

# NWGetActiveProtocolStacks

Returns protocol stack information in NWFSE_ACTIVE_STACKS.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetActiveProtocolStacks
  (NWCONN_HANDLE                conn,
   nuint32                      startNum,
   NWFSE_ACTIVE_STACKS  N_FAR   *fseActiveStacks);
```

## Delphi Syntax

```
uses calwin32

Function NWGetActiveProtocolStacks
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   Var fseActiveStacks : NWFSE_ACTIVE_STACKS
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the number to start with if NWGetActiveProtocolStacks is called iteratively.

**fseActiveStacks**

(OUT) Points to NWFSE_ACTIVE_STACKS.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FF | Failure or Invalid Start Number |

## Remarks

To call NWGetActiveProtocolStacks, you must have console operator rights.

In the first call, `startNumber` should be 0. On subsequent calls, use the number of stacks retrieved.

## NCP Calls

0x2222 123 40 Active Protocol Stacks

## See Also

NWGetProtocolStackConfigInfo (page 133), NWGetProtocolStackCustomInfo (page 135), NWGetProtocolStackStatsInfo (page 137), NWGetProtocolStkNumsByLANBrdNum (page 139), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetCacheInfo

Allows a caller from a workstation to get server cache management statistical and operating system information.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetCacheInfo  (
   NWCONN_HANDLE              conn,
   NWFSE_CACHE_INFO  N_FAR   *fseCacheInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetCacheInfo
  (conn : NWCONN_HANDLE;
   Var fseCacheInfo : NWFSE_CACHE_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseCacheInfo**

(IN) Points to NWFSE_CACHE_INFO.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetCacheInfo, you must have console operator rights.

## NCP Calls

0x2222 123 01 Get Cache Information

## See Also

NWGetDirCacheInfo (page 55)

# NWGetCPUInfo

Gets CPU and hardware configuration information about the 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetCPUInfo  (
   NWCONN_HANDLE            conn,
   nuint32                  CPUNum,
   pnstr8                   CPUName,
   pnstr8                   numCoprocessor,
   pnstr8                   bus,
   NWFSE_CPU_INFO  N_FAR   *fseCPUInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetCPUInfo
  (conn : NWCONN_HANDLE;
   CPUNum : nuint32;
   CPUName : pnstr8;
   numCoprocessor : pnstr8;
   bus : pnstr8;
   Var fseCPUInfo : NWFSE_CPU_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**CPUNum**

(IN) Specifies the CPU number. Pass one for NetWare 5.x and 6.x and zero for NetWare 4.0.

**CPUName**

(OUT) Points to the ASCII string of the CPU type. This string space must be allocated by the application.

**numCoprocessor**

(OUT) Points to the ASCII string of whether or not a coprocessor is present. This string space must be allocated by the application.

**bus**

(OUT) Points to the ASCII string of the bus type. This string space must be allocated by the application.

**fseCPUInfo**

(OUT) Points to NWFSE_CPU_INFO.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x89FE | DIRECTORY_LOCKED |
| 0x89FF | Failure or Invalid CPU Number |

## Remarks

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

Console operator rights are NOT necessary to call NWGetCPUInfo.

## NCP Calls

0x2222 123 08 CPU Information

# NWGetDirCacheInfo

Returns information about the directory cache manager.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetDirCacheInfo  (
   NWCONN_HANDLE                   conn,
   NWFSE_DIR_CACHE_INFO  N_FAR   *fseDirCacheInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetDirCacheInfo
  (conn : NWCONN_HANDLE;
   Var fseDirCacheInfo : NWFSE_DIR_CACHE_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseDirCacheInfo**

(OUT) Points to NWFSE_DIR_CACHE_INFO.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetDirCacheInfo, you must have console operator rights.

## NCP Calls

0x2222 123 12 Get Directory Cache Information

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

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

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

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

# NWGetFileServerDateAndTime

Returns the network date and time maintained 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetFileServerDateAndTime  (
   NWCONN_HANDLE   conn,
   pnuint8         dateTimeBuffer);
```

## Delphi Syntax

```
uses calwin32

Function NWGetFileServerDateAndTime
  (conn : NWCONN_HANDLE;
   dateTimeBuffer : pnuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**dateTimeBuffer**

(OUT) Points to a 7-byte buffer for the network date and time.

## 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

Since the date and time are not automatically synchronized across an internetwork, dates and times may differ among servers.

The system time clock is a 7-byte value defined in the following format:

| Byte | Value | Range |
|---|---|---|
| 0 | Year | 80 through 179 |
| 1 | Month | 1 through 12 |
| 2 | Day | 1 through 31 |
| 3 | Hour | 0 through 23 (0 = 12 midnight; 23 = 11 PM) |
| 4 | Minute | 0 through 59 |
| 5 | Second | 0 through 59 |
| 6 | Day of Week | 0 through 6, 0=Sunday |

**NOTE:** The year value corresponds to the specified years:

80-99 1980-1999
100-179 2000-2079

## NCP Calls

0x2222 20 Get File Server Date And Time

# NWGetFileServerDescription

Returns information about a NetWare server, including company name, NetWare version, revision date and copyright notice, using descriptive strings.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetFileServerDescription  (
   NWCONN_HANDLE   conn,
   pnstr8          companyName,
   pnstr8          revision,
   pnstr8          revisionDate,
   pnstr8          copyrightNotice);
```

## Delphi Syntax

```
uses calwin32

Function NWGetFileServerDescription
  (conn : NWCONN_HANDLE;
   companyName : pnstr8;
   revision : pnstr8;
   revisionDate : pnstr8;
   copyrightNotice : pnstr8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**companyName**

(OUT) Points to the name of the company providing the version of NetWare (80 characters, optional).

**revision**

(OUT) Points to the NetWare version and revision description string (80 characters, optional).

**revisionDate**

(OUT) Points to the revision date in the form xx/xx/xx. For example: 12/16/91 (24 characters, optional).

**copyrightNotice**

(OUT) Points to the copyright notice (80 characters, 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 |

## Remarks

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

Each string is NULL-terminated.

For items not desired in the return, substitute NULL. However, all parameter positions must be filled.

Any client attached to the specified server can call NWGetFileServerDescription. Console operator rights are not required.

## NCP Calls

0x2222 23 201 Get File Server Description Strings

# NWGetFileServerExtendedInfo

Returns extended information about the specified NetWare server, including versions for accounting, VAP, queueing, print server, virtual console, security and internet bridging.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetFileServerExtendedInfo  (
   NWCONN_HANDLE   conn,
   pnuint8         accountingVer,
   pnuint8         VAPVer,
   pnuint8         queueingVer,
   pnuint8         printServerVer,
   pnuint8         virtualConsoleVer,
   pnuint8         securityVer,
   pnuint8         internetBridgeVer);
```

## Delphi Syntax

```
uses calwin32

Function NWGetFileServerExtendedInfo
  (conn : NWCONN_HANDLE;
   accountingVer : pnuint8;
   VAPVer : pnuint8;
   queueingVer : pnuint8;
   printServerVer : pnuint8;
   virtualConsoleVer : pnuint8;
   securityVer : pnuint8;
   internetBridgeVer : pnuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**accountingVer**

(OUT) Points to the accounting version number (optional).

**VAPVer**

(OUT) Points to the VAP version number (optional).

**queueingVer**

(OUT) Points to the queueing version number (optional).

**printServerVer**

(OUT) Points to the print server version number (optional).

**virtualConsoleVer**

(OUT) Points to the virtual console version number (optional).

**securityVer**

(OUT) Points to the security version number (optional).

**internetBridgeVer**

(OUT) Points to the internet bridging version number (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 |

## Remarks

NWGetFileServerExtendedInfo returns values as a single byte per parameter, individual values between 0 and 255.

If you don't want certain information, substitute NULL. However, all parameter positions must be filled.

To call NWGetFileServerExtendedInfo, you must have console operator rights.

## NCP Calls

0x2222 23 17 Get File Server Information

## See Also

NWGetFileServerInformation (page 67), NWGetFileServerVersionInfo (page 74)

# NWGetFileServerInfo

Calls for the server's operational statistics.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetFileServerInfo  (
   NWCONN_HANDLE                    conn,
   NWFSE_FILE_SERVER_INFO  N_FAR   *fseFileServerInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetFileServerInfo
  (conn : NWCONN_HANDLE;
   Var fseFileServerInfo : NWFSE_FILE_SERVER_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseFileServerInfo**

(OUT) Points to NWFSE_FILE_SERVER_INFO to get NetWare server information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

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

Console operator rights are NOT necessary to call NWGetFileServerInfo.

## NCP Calls

0x2222 123 02 Get File Server Information

# NWGetFileServerInformation

Returns several items, including NetWare server name, NetWare versions, maximum and peak connections, number of licensed connections currently in use, maximum volumes supported, and SFT and TTS level of support.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetFileServerInformation  (
   NWCONN_HANDLE   conn,
   pnstr8          serverName,
   pnuint8         majorVer,
   pnuint8         minVer,
   pnuint8         rev,
   pnuint16        maxConns,
   pnuint16        maxConnsUsed,
   pnuint16        connsInUse,
   pnuint16        numVolumes,
   pnuint8         SFTLevel,
   pnuint8         TTSLevel);
```

## Delphi Syntax

```
uses calwin32

Function NWGetFileServerInformation
  (conn : NWCONN_HANDLE;
   serverName : pnstr8;
   majorVer : pnuint8;
   minVer : pnuint8;
   rev : pnuint8;
   maxConns : pnuint16;
   maxConnsUsed : pnuint16;
   connsInUse : pnuint16;
   numVolumes : pnuint16;
   SFTLevel : pnuint8;
   TTSLevel : pnuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**serverName**

(OUT) Points to the name of NetWare server (48 bytes, optional).

**majorVer**

(OUT) Points to the major NetWare version number (optional).

**minVer**

(OUT) Points to the minor NetWare version number (optional).

**rev**

(OUT) Points to the revision number of the NetWare OS on NetWare server (optional).

**maxConns**

(OUT) Points to the maximum number of connections the server will support (optional). The connection table for NetWare 4.x is dynamic. This number will be the maximum of what the table has grown to.

**maxConnsUsed**

(OUT) Points to the highest number of connections simultaneously in use (optional).

**connsInUse**

(OUT) Points to the number of licensed connections the server currently has in use (optional).

**numVolumes**

(OUT) Points to the maximum number of volumes the server will support (optional).

**SFTLevel**

(OUT) Points to the SFT level the server supports (optional).

**TTSLevel**

(OUT) Points to the TTS Level of NetWare server operating system (optional).

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8996 | SERVER_OUT_OF_MEMORY |

## Remarks

NWGetFileServerInformation will only return the number of licensed connections in use. For NetWare 3.x, all connections are considered licensed. For NetWare 4.x, many connections do not require a server license and will not be returned in `connsInUse`.

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

The buffer allocated to receive NetWare server name should be at least 48 bytes long.

Substitute NULL for a parameter if a return is not desired. However, all parameter positions must be filled. Any client can call NWGetFileServerInformation without logging in to the specified NetWare server.

## NCP Calls

0x2222 23 17 Get File Server Information

## See Also

NWGetFileServerInformation (page 67), NWGetFileServerVersionInfo (page 74)

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

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

# NWGetFileServerLoginStatus

Returns a NetWare server's login status.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetFileServerLoginStatus  (
   NWCONN_HANDLE   conn,
   pnuint8         loginEnabledFlag);
```

## Delphi Syntax

```
uses calwin32

Function NWGetFileServerLoginStatus
  (conn : NWCONN_HANDLE;
   loginEnabledFlag : pnuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**loginEnabledFlag**

(OUT) Points to a zero flag if clients cannot log in and non-zero if they can.

## 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 |

## Remarks

NWGetFileServerLoginStatus determines if the users' logins are currently allowed on the target server.

## NCP Calls

0x2222 23 205 Get File Server Login Status

# NWGetFileServerMiscInfo (obsolete 12/98)

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

# NWGetFileServerVersionInfo

Returns information about a NetWare server's name and version levels.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetFileServerVersionInfo  (
   NWCONN_HANDLE          conn,
   VERSION_INFO  N_FAR   *versBuffer);
```

## Delphi Syntax

```
uses calwin32

Function NWGetFileServerVersionInfo
  (conn : NWCONN_HANDLE;
   Var versBuffer : VERSION_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**versBuffer**

(OUT) Points to the VERSION_INFO structure, which contains the NetWare server version information.

## 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 |

## Remarks

Console operator rights and an authenticated connection are NOT necessary to call NWGetFileServerVersionInfo; you only need an attachment.

To get product version information (major, minor, and revision), call NWGetNetWareProductVersion (page 114).

## NCP Calls

0x2222 23 17 Get File Server Information

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

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

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

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

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

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

# NWGetGarbageCollectionInfo

Returns the server's memory manager's statistical information.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetGarbageCollectionInfo  (
   NWCONN_HANDLE                           conn,
   NWFSE_GARBAGE_COLLECTION_INFO  N_FAR   *fseGarbageCollectionInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetGarbageCollectionInfo
  (conn : NWCONN_HANDLE;
   Var fseGarbageCollectionInfo : NWFSE_GARBAGE_COLLECTION_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseGarbageCollectionInfo**

(OUT) Points to NWFSE_GARBAGE_COLLECTION_INFO returning garbage collection information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetGarbageCollectionInfo, you must have console operator rights.

## NCP Calls

0x2222 123 07 Garbage Collection Information

# NWGetGeneralRouterAndSAPInfo

Returns router information received from RIP and SAP packets.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetGeneralRouterAndSAPInfo  (
   NWCONN_HANDLE                              conn,
   NWFSE_GENERAL_ROUTER_SAP_INFO  N_FAR   *fseGeneralRouterSAPInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetGeneralRouterAndSAPInfo
  (conn : NWCONN_HANDLE;
   Var fseGeneralRouterSAPInfo : NWFSE_GENERAL_ROUTER_SAP_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseGeneralRouterSAPInfo**

(OUT) Points to NWFSE_GENERAL_ROUTER_SAP_INFO returning general router SAP information.

## 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 |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

## Remarks

To call NWGetGeneralRouterAndSAPInfo, you must have console operator rights.

## NCP Calls

0x2222 123 50 Get General Router and SAP Information

# NWGetIPXSPXInfo

Returns the server's internal IPX and SPX statistics information.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetIPXSPXInfo  (
   NWCONN_HANDLE               conn,
   NWFSE_IPXSPX_INFO  N_FAR   *fseIPXSPXInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetIPXSPXInfo
  (conn : NWCONN_HANDLE;
   Var fseIPXSPXInfo : NWFSE_IPXSPX_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseIPXSPXInfo**

(OUT) Points to NWFSE_IPXSPX_INFO returning IPX/SPX information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

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

Console operator rights are NOT necessary to call NWGetIPXSPXInfo.

## NCP Calls

0x2222 123 06 IPX SPX Information

# NWGetKnownNetworksInfo

Returns information about networks for which the server has received Routing Information Packets (RIPs).

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetKnownNetworksInfo
  (NWCONN_HANDLE                        conn,
   nuint32                              startNum,
   NWFSE_KNOWN_NETWORKS_INFO  N_FAR   *fseKnownNetworksInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetKnownNetworksInfo
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   Var fseKnownNetworksInfo : NWFSE_KNOWN_NETWORKS_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the starting network with which to begin the search (commonly 0 to begin the search, on subsequent calls it should be the total number of networks returned up to the call.)

**fseKnownNetworksInfo**

(OUT) Points to NWFSE_KNOWN_NETWORKS_INFO returning information about known networks.

## 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 |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetKnownNetworksInfo, you must have console operator rights.

## NCP Calls

0x2222 123 53 Get Known Networks Information

## See Also

NWGetNetworkRouterInfo (page 116), NWGetNetworkRoutersInfo (page 118)

# NWGetKnownServersInfo

Returns information about servers advertising themselves to the server with SAP packets.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetKnownServersInfo
  (NWCONN_HANDLE                    conn,
   nuint32                          startNum,
   nuint32                          serverType,
   NWFSE_KNOWN_SERVER_INFO  N_FAR   *fseKnownServerInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetKnownServersInfo
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   serverType : nuint32;
   Var fseKnownServerInfo : NWFSE_KNOWN_SERVER_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the cumulative number of servers returned from all previous calls; normally, zero (0) for the first call.

**serverType**

(IN) Specifies the server type:

0x0400 NetWare server
0xFFFF All other server types

**fseKnownServerInfo**

(OUT) Points to NWFSE_KNOWN_SERVER_INFO containing known server information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x8901 | Returned from NWFSE_KNOWN_SERVER_INFO when no more items are found |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

## Remarks

To call NWGetKnownServersInfo, you must have console operator rights.

`startNum` should be set to zero on the first call. On subsequent calls, the value returned in the `numberOfEntries` field in the SERVER_AND_VCONSOLE_INFO structure should be added to the value in `startNum` until INVALID_CONNECTION is returned.

## NCP Calls

0x2222 123 56 Get Known Servers Information

## See Also

NWGetServerSourcesInfo (page 153)

# NWGetLANCommonCountersInfo

Returns common statistics for a LAN board.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetLANCommonCountersInfo  (
   NWCONN_HANDLE                           conn,
   nuint32                                 boardNum,
   nuint32                                 blockNum,
   NWFSE_LAN_COMMON_COUNTERS_INFO N_FAR   *fseLANCommonCountersInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetLANCommonCountersInfo
  (conn : NWCONN_HANDLE;
   boardNum : nuint32;
   blockNum : nuint32;
   Var fseLANCommonCountersInfo : NWFSE_LAN_COMMON_COUNTERS_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**boardNum**

(IN) Specifies the board numbers returned by NWGetActiveLANBoardList.

**blockNum**

(IN) Specifies the starting number of the common counters to return; usually set to zero (0) to return all the counters.

**fseLANCommonCountersInfo**

(OUT) Points to NWFSE_LAN_COMMON_COUNTERS_INFO returning LAN common counters information.

## 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 |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FF | Failure or Invalid Board or Block Number |

## Remarks

To call NWGetLANCommonCountersInfo, you must have console operator rights.

## NCP Calls

0x2222 123 22 LAN Common Counters Information

## See Also

NWGetActiveLANBoardList (page 47), NWGetLANConfigInfo (page 91), NWGetLANCustomCountersInfo (page 93), NWGetLSLLogicalBoardStats (page 99), NWGetProtocolStkNumsByLANBrdNum (page 139)

# NWGetLANConfigInfo

Returns configuration information for the LAN board identified by `boardNum` .

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetLANConfigInfo
  (NWCONN_HANDLE                    conn,
   nuint32                          boardNum,
   NWFSE_LAN_CONFIG_INFO  N_FAR    *fseLANConfigInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetLANConfigInfo
  (conn : NWCONN_HANDLE;
   boardNum : nuint32;
   Var fseLANConfigInfo : NWFSE_LAN_CONFIG_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**boardNum**

(IN) Specifies the number of the LAN board for which you want LAN driver information.

**fseLANConfigInfo**

(OUT) Points to NWFSE_LAN_CONFIG_INFO returning LAN configuration information.

## 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 |
| 0x8979 | ERR_NO_ITEMS_FOUND |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FF | Failure or Invalid Board Number |

## Remarks

To call NWGetLANConfigInfo, you must have console operator rights.

## NCP Calls

0x2222 123 21 LAN Configuration Information

## See Also

NWGetActiveLANBoardList (page 47), NWGetLANCommonCountersInfo (page 89), NWGetLANCustomCountersInfo (page 93), NWGetLSLLogicalBoardStats (page 99), NWGetProtocolStkNumsByLANBrdNum (page 139)

# NWGetLANCustomCountersInfo

Returns custom statistics for a LAN board.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetLANCustomCountersInfo
  (NWCONN_HANDLE                    conn,
   nuint32                          boardNum,
   nuint32                          startingNum,
   NWFSE_LAN_CUSTOM_INFO  N_FAR    *fseLANCustomInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetLANCustomCountersInfo
  (conn : NWCONN_HANDLE;
   boardNum : nuint32;
   startingNum : nuint32;
   Var fseLANCustomInfo : NWFSE_LAN_CUSTOM_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**boardNum**

(IN) Specifies the board number returned by NWGetActiveLANBoardList.

**startingNum**

(IN) Specifies the cumulative number of custom counters already returned; normally, zero (0) for the first call.

**fseLANCustomInfo**

(OUT) Points to NWFSE_LAN_CUSTOM_INFO returning information.

## 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 |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FF | Failure or Invalid Start or Board Number |

## Remarks

To call NWGetLANCustomCountersInfo, you must have console operator rights.

## NCP Calls

0x2222 123 23 LAN Custom Counters Information

## See Also

NWGetActiveLANBoardList (page 47), NWGetLANConfigInfo (page 91), NWGetLANCommonCountersInfo (page 89), NWGetLSLLogicalBoardStats (page 99), NWGetProtocolStkNumsByLANBrdNum (page 139)

# NWGetLoadedMediaNumList

Returns a list of loaded media numbers.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetLoadedMediaNumList  (
   NWCONN_HANDLE                         conn,
   NWFSE_LOADED_MEDIA_NUM_LIST  N_FAR   *fseLoadedMediaNumList);
```

## Delphi Syntax

```
uses calwin32

Function NWGetLoadedMediaNumList
  (conn : NWCONN_HANDLE;
   Var fseLoadedMediaNumList : NWFSE_LOADED_MEDIA_NUM_LIST
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseLoadedMediaNumList**

(OUT) Points to NWFSE_LOADED_MEDIA_NUM_LIST returning information.

## 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 |

| | |
|---|---|
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

## Remarks

To call NWGetLoadedMediaNumList, you must have console operator rights.

## NCP Calls

0x2222 123 47 Get Loaded Media Num List

## See Also

NWGetMediaMgrObjChildrenList (page 101), NWGetMediaMgrObjInfo (page 103), NWGetMediaMgrObjList (page 105), NWGetMediaNameByMediaNum (page 108), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetLSLInfo

Returns LSL information.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetLSLInfo  (
   NWCONN_HANDLE            conn,
   NWFSE_LSL_INFO  N_FAR   *fseLSLInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetLSLInfo
  (conn : NWCONN_HANDLE;
   Var fseLSLInfo : NWFSE_LSL_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseLSLInfo**

(OUT) Points to NWFSE_LSL_INFO returning LSL™ information.

## 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 |

| | |
|---|---|
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

## Remarks

To call NWGetLSLInfo, you must have console operator rights.

## NCP Calls

0x2222 123 25 LSL Information

## See Also

NWGetLSLLogicalBoardStats (page 99)

# NWGetLSLLogicalBoardStats

Returns LSL logical board statistics.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetLSLLogicalBoardStats (
   NWCONN_HANDLE                          conn,
   nuint32                                LANBoardNum,
   NWFSE_LSL_LOGICAL_BOARD_STATS  N_FAR   *fseLSLLogicalBoardStats);
```

## Delphi Syntax

```
uses calwin32

Function NWGetLSLLogicalBoardStats
  (conn : NWCONN_HANDLE;
   LANBoardNum : nuint32;
   Var fseLSLLogicalBoardStats : NWFSE_LSL_LOGICAL_BOARD_STATS
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**LANBoardNum**

(IN) Specifies a board number returned by NWGetActiveLANBoardList.

**fseLSLLogicalBoardStats**

(OUT) Points to NWFSE_LSL_LOGICAL_BOARD_STATS returning LSL logical board statistics.

## Return Values

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

| | |
|---|---|
| 0x00000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FB | ERR_NCP_NOT_SUPPORTED |
| 0x89FF | Failure or Invalid LAN Board Number |

## Remarks

To call NWGetLSLLogicalBoardStats, you must have console operator rights.

## NCP Calls

0x2222 123 26 LSL Logical Board Statistics

## See Also

NWGetLSLInfo (page 97)

# NWGetMediaMgrObjChildrenList

Returns a list of children belonging to a given media manager parent object.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetMediaMgrObjChildrenList  (
   NWCONN_HANDLE                       conn,
   nuint32                             startNum,
   nuint32                             objType,
   nuint32                             parentObjNum,
   NWFSE_MEDIA_MGR_OBJ_LIST  N_FAR   *fseMediaMgrObjList);
```

## Delphi Syntax

```
uses calwin32

Function NWGetMediaMgrObjChildrenList
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   objType : nuint32;
   parentObjNum : nuint32;
   Var fseMediaMgrObjList : NWFSE_MEDIA_MGR_OBJ_LIST
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the value returned in `nextStartObjNum` of NWFSE_MEDIA_MGR_OBJ_LIST; Normally, zero (0) on the first call.

**objType**

(IN) Specifies one of the types defined in nwfse.h, such as ADAPTER_OBJECT or MIRROR_OBJECT.

**parentObjNum**

(IN) Specifies the parent object ID number such as one returned by NWGetMediaMgrObjList.

**fseMediaMgrObjList**

(OUT) Points to NWFSE_MEDIA_MGR_OBJ_LIST listing media manager object's children.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8979 | Invalid Start Number, Object Type, or Parent Object Number |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetMediaMgrObjChildrenList, you must have console operator rights.

## NCP Calls

0x2222 123 32 Get Media Manager Object Children's List

## See Also

NWGetLoadedMediaNumList (page 95), NWGetMediaMgrObjInfo (page 103), NWGetMediaMgrObjList (page 105), NWGetMediaNameByMediaNum (page 108), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetMediaMgrObjInfo

Returns information about media manager objects (storage devices).

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetMediaMgrObjInfo  (
   NWCONN_HANDLE                      conn,
   nuint32                            objNum,
   NWFSE_MEDIA_MGR_OBJ_INFO  N_FAR   *fseMediaMgrObjInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetMediaMgrObjInfo
  (conn : NWCONN_HANDLE;
   objNum : nuint32;
   Var fseMediaMgrObjInfo : NWFSE_MEDIA_MGR_OBJ_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**objNum**

(IN) Specifies the object ID number returned by NWGetMediaMgrObjList representing the device you want information about.

**fseMediaMgrObjInfo**

(OUT) Points to NWFSE_MEDIA_MGR_OBJ_INFO returning media manager object information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetMediaMgrObjInfo, you must have console operator rights.

Media manager objects are storage devices which are managed by the OS in an object-oriented database to allow for the needs of current and future file system applications and storage applications.

## NCP Calls

0x2222 123 30 Get Media Manager Object Information

## See Also

NWGetLoadedMediaNumList (page 95), NWGetMediaMgrObjChildrenList (page 101), NWGetMediaMgrObjList (page 105), NWGetMediaNameByMediaNum (page 108), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetMediaMgrObjList

Returns a list of media manager objects (storage devices) of the specified type.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetMediaMgrObjList  (
   NWCONN_HANDLE                      conn,
   nuint32                            startNum,
   nuint32                            objType,
   NWFSE_MEDIA_MGR_OBJ_LIST  N_FAR   *fseMediaMgrObjList);
```

## Delphi Syntax

```
uses calwin32

Function NWGetMediaMgrObjList
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   objType : nuint32;
   Var fseMediaMgrObjList : NWFSE_MEDIA_MGR_OBJ_LIST
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the value returned in the `nextStartObjNum` field of the `fseMediaMgrObjList` parameter (set to -1 initially).

**objType**

(IN) Specifies the object type.

**fseMediaMgrObjList**

(OUT) Points to the NWFSE_MEDIA_MGR_OBJ_LIST structure listing media manager objects.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8979 | Invalid Start Number or Object Type |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetMediaMgrObjList, you must have console operator rights.

Media manager objects are storage devices maintained by the media manager in an object-oriented database.

Object types are the following:

| C Value | Delphi Value | Value Name |
|---|---|---|
| 0 | 0 | FSE_ADAPTER_OBJECT |
| 1 | 1 | FSE_CHANGER_OBJECT |
| 2 | 2 | FSE_DEVICE_OBJECT |
| 4 | 4 | FSE_MEDIA_OBJECT |
| 5 | 5 | FSE_PARTITION_OBJECT |
| 6 | 6 | FSE_SLOT_OBJECT |
| 7 | 7 | FSE_HOTFIX_OBJECT |
| 8 | 8 | FSE_MIRROR_OBJECT |
| 9 | 9 | FSE_PARITY_OBJECT |
| 10 | 10 | FSE_VOLUME_SEG_OBJECT |
| 11 | 11 | FSE_VOLUME_OBJECT |
| 12 | 12 | FSE_CLONE_OBJECT |
| 14 | 14 | FSE_MAGAZINE_OBJECT |
| 15 | 15 | FSE_VIRTUAL_DEVICE_OBJECT |
| 0xFFFF | $FFFF | FSE_UNKNOWN_OBJECT |

## NCP Calls

0x2222 123 31 Get Media Manager Objects List

## See Also

NWGetLoadedMediaNumList (page 95), NWGetMediaMgrObjInfo (page 103), NWGetMediaMgrObjChildrenList (page 101), NWGetMediaNameByMediaNum (page 108), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetMediaNameByMediaNum

Returns the identifying name or label for the given media object.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetMediaNameByMediaNum
  (NWCONN_HANDLE                    conn,
   nuint32                          mediaNum,
   pnstr8                           mediaName,
   NWFSE_MEDIA_NAME_LIST  N_FAR    *fseMediaNameList);
```

## Delphi Syntax

```
uses calwin32

Function NWGetMediaNameByMediaNum
  (conn : NWCONN_HANDLE;
   mediaNum : nuint32;
   mediaName : pnstr8;
   Var fseMediaNameList : NWFSE_MEDIA_NAME_LIST
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**mediaNum**

(IN) Specifies the object ID number of the target media object returned by calling either NWGetMediaMgrObjList or NWGetMediaMgrChildrenList.

**mediaName**

(OUT) Points to the name of the media object specified by `mediaNum`.

**fseMediaNameList**

(OUT) Points to NWFSE_MEDIA_NAME_LIST returning information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetMediaNameByMediaNum, you must have console operator rights.

`mediaName` requires a buffer at least as large as FSE_MEDIA_NAME_LEN_MAX +1. It can be longer than that if desired.

## NCP Calls

0x2222 123 46 Get Media Name By Media Number

## See Also

NWGetLoadedMediaNumList (page 95), NWGetMediaMgrObjInfo (page 103), NWGetMediaMgrObjChildrenList (page 101), NWGetMediaMgrObjList (page 105), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetMLIDBoardInfo

Returns a list of and information for each Multiple Link Interface Driver (MLID) bound to a network board.

**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:** Server Environment

## Syntax

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

NWCCODE NWGetMLIDBoardInfo (
   NWCONN_HANDLE                  conn,
   nuint32                        MLIDBoardNum,
   NWFSE_MLID_BOARD_INFO N_FAR    *fseMLIDBoardInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetMLIDBoardInfo
   (conn : NWCONN_HANDLE;
   MLIDBoardNum : nuint32;
   var fseMLIDBoardInfo : NWFSE_MLID_BOARD_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**MLIDBoardNum**

(IN) Specifies one of the board numbers returned by calling NWGetActiveLANBoardList.

**fseMLIDBoardInfo**

(OUT) Points to NWFSE_MLID_BOARD_INFO, which contains the MLID list and information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0c89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FF | Failure or Invalid LAN Board Number |

## Remarks

You must have CONSOLE privileges to call NWGetMLIDBoardInfo.

NWGetMLIDBoardInfo returns 0x89FF if no procotols are bound to the specified board.

NetWare 4.x returns only one protocol per board.

**IMPORTANT:** Some older versions of the Win32 DLLs do not include this function.

## NCP Calls

0x2222 123 27 MLID Board Information

## See Also

NWGetActiveLANBoardList (page 47)

# NWGetNetWareFileSystemsInfo

Returns information about a loaded file system.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetNetWareFileSystemsInfo  (
   NWCONN_HANDLE                    conn,
   NWFSE_FILE_SYSTEM_INFO  N_FAR   *fseFileSystemInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetNetWareFileSystemsInfo
  (conn : NWCONN_HANDLE;
   Var fseFileSystemInfo : NWFSE_FILE_SYSTEM_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseFilesystemInfo**

(OUT) Points to NWFSE_FILE_SYSTEMS_INFO returning NetWare file systems information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetNetWareFileSystemsInfo, you must have console operator rights.

## NCP Calls

0x2222 123 03 NetWare File Systems Information

# NWGetNetWareProductVersion

Returns the NetWare product version.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetNetWareProductVersion  (
   NWCONN_HANDLE                       conn,
   NETWARE_PRODUCT_VERSION  N_FAR   *version);
```

## Delphi Syntax

```
uses calwin32

Function NWGetNetWareProductVersion
  (conn : NWCONN_HANDLE;
   Var version : NETWARE_PRODUCT_VERSION
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**version**

(OUT) Points to the NETWARE_PRODUCT_VERSION structure, which contains the NetWare product version information.

## 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 |
| 0x8996 | SERVER_OUT_OF_MEMORY |

## Remarks

Console operator rights and an authenticated connection are NOT necessary to call NWGetNetWareProductVersion; you only need an attachment.

NWGetNetWareProductVersion supports all versions of NetWare servers, and is run on all supported platforms. Prior to NetWare 5.1, the version returned will be the OS version or the server version; however, starting with NetWare 5.1 these might not be the same because the OS does not always change with the release of a new product version of NetWare.

If you use the version command on the server console, the Novell NetWare version returned is equivalent to the product version (NetWare 5.1 and greater) or the OS version (prior to NetWare 5.1) returned from the NWGetNetWareProductVersion call. The Server Version returned from the version command on the server console is equivalent to the server version returned from the NWGetFileServerVersionInfo (page 74) call or the OS version returned from the NWGetOSVersionInfo (page 128) call. The server version and the OS version are the same.

## NCP Calls

0x2222 23 17 Get File Server Information

# NWGetNetworkRouterInfo

Returns information about network routing (other networks known to NetWare 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetNetworkRouterInfo  (
   NWCONN_HANDLE                       conn,
   nuint32                             networkNum,
   NWFSE_NETWORK_ROUTER_INFO  N_FAR   *fseNetworkRouterInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetNetworkRouterInfo
  (conn : NWCONN_HANDLE;
   networkNum : nuint32;
   Var fseNetworkRouterInfo : NWFSE_NETWORK_ROUTER_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**networkNum**

(IN) Specifies the number of the network for which to return information.

**fseNetworkRouterInfo**

(OUT) Points to NWFSE_NETWORK_ROUTER_INFO returning network router information.

## 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

To call NWGetNetworkRouterInfo, you must have console operator rights.

## NCP Calls

0x2222 123 51 Get Network Router Information

## See Also

NWGetKnownNetworksInfo (page 85), NWGetNetworkRoutersInfo (page 118)

# NWGetNetworkRoutersInfo

Returns information about the routers on a network.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetNetworkRoutersInfo  (
   NWCONN_HANDLE                        conn,
   nuint32                              networkNum,
   nuint32                              startNum,
   NWFSE_NETWORK_ROUTERS_INFO  N_FAR   *fseNetworkRoutersInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetNetworkRoutersInfo
  (conn : NWCONN_HANDLE;
   networkNum : nuint32;
   startNum : nuint32;
   Var fseNetworkRoutersInfo : NWFSE_NETWORK_ROUTERS_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**networkNum**

(IN) Specifies the number of the network for which to return information.

**startNum**

(IN) Specifies the value returned in `numberOfEntries` in NWFSE_NETWORK_ROUTERS_INFO; normally, zero (0) on the first call.

**fseNetworkRoutersInfo**

(OUT) Points to NWFSE_NETWORK_ROUTERS_INFO returning network routers information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | Invalid Network Number or Start Number |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetNetworkRoutersInfo, you must have console operator rights.

## NCP Calls

0x2222 123 52 Get Network Routers Information

## See Also

NWGetKnownNetworksInfo (page 85), NWGetNetworkRouterInfo (page 116)

# NWGetNetworkSerialNumber

Returns the NetWare server's serial number and the application number.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetNetworkSerialNumber  (
   NWCONN_HANDLE   conn,
   pnuint32        serialNum,
   pnuint16        appNum);
```

## Delphi Syntax

```
uses calwin32

Function NWGetNetworkSerialNumber
  (conn : NWCONN_HANDLE;
   serialNum : pnuint32;
   appNum : pnuint16
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**serialNum**

(OUT) Points to the NetWare server's serial number.

**appNum**

(OUT) Points to the application number.

## 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 |

## Remarks

The combination of the server serial number and the application number is unique.

## NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 18 Get Network Serial Number

# NWGetNLMInfo

Returns information about a specific loaded NLM.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetNLMInfo  (
   NWCONN_HANDLE            conn,
   nuint32                  NLMNum,
   pnstr8                   fileName,
   pnstr8                   NLMname,
   pnstr8                   copyright,
   NWFSE_NLM_INFO  N_FAR   *fseNLMInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetNLMInfo
  (conn : NWCONN_HANDLE;
   NLMNum : nuint32;
   fileName : pnstr8;
   NLMname : pnstr8;
   copyright : pnstr8;
   Var fseNLMInfo : NWFSE_NLM_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**NLMNum**

(IN) Specifies the NLM ID number returned by NWGetNLMLoadedList.

**fileName**

(OUT) Points to the name of the NLM file.

**NLMName**

(OUT) Points to a short description of the NLM.

**copyright**

(OUT) Points to the copyright string (optional).

**fseNLMInfo**

(OUT) Points to NWFSE_NLM_INFO.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FF | Failure or Invalid NLM Number |

## Remarks

To call NWGetNLMInfo, you must have console operator rights.

NWGetNLMInfo should only be called for NLMs that have information in the buffer.

## NCP Calls

0x2222 123 11 NLM Information

## See Also

NWGetNLMLoadedList (page 124), NWGetNLMsResourceTagList (page 126)

# NWGetNLMLoadedList

Returns a list of loaded NLM sequence numbers.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetNLMLoadedList  (
   NWCONN_HANDLE                    conn,
   nuint32                          startNum,
   NWFSE_NLM_LOADED_LIST  N_FAR   *fseNLMLoadedList);
```

## Delphi Syntax

```
uses calwin32

Function NWGetNLMLoadedList
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   Var fseNLMLoadedList : NWFSE_NLM_LOADED_LIST
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the starting number (set to zero the first time NWGetNLMLoadedList is called).

**fseNLMLoadedList**

(OUT) Points to the NWFSE_NLM_LOADED_LIST structure.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FF | Failure |

## Remarks

To call NWGetNLMLoadedList, you must have console operator rights.

NWGetNLMLoadedList will only return information about 130 NLMs at a time. If you have more than 130 NLMs loaded, call NWGetNLMLoadedList iteratively and pass to `startNum` the number of the next NLM that you want to start the list with. For example:

```
fseNLMLoadedList->NLMNums[fseNLMLoadedList->NLMsInList]+1
```

## NCP Calls

0x2222 123 10 Get NLM Loaded List

## See Also

NWGetNLMInfo (page 122), NWGetNLMsResourceTagList (page 126)

# NWGetNLMsResourceTagList

Returns information about resources used by NLMs on a 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetNLMsResourceTagList  (
   NWCONN_HANDLE                          conn,
   nuint32                                NLMNum,
   nuint32                                startNum,
   NWFSE_NLMS_RESOURCE_TAG_LIST  N_FAR   *fseNLMsResourceTagList);
```

## Delphi Syntax

```
uses calwin32

Function NWGetNLMsResourceTagList
  (conn : NWCONN_HANDLE;
   NLMNum : nuint32;
   startNum : nuint32;
   Var fseNLMsResourceTagList : NWFSE_NLMS_RESOURCE_TAG_LIST
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**NLMNum**

(IN) Specifies the NLM ID number representing the NLM on the server, returned by NWGetNLMLoadedList.

**startNum**

(IN) Specifies the previous `startNum`, plus the value in `packetResourceTags` of NWFSE_NLMS_RESOURCE_TAG_LIST; normally 0 (zero) on the first call.

**fseNLMsResourceTagList**

(OUT) Points to NWFSE_NLMS_RESOURCE_TAG_LIST.

## Return Values

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

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

## Remarks

To call NWGetNLMsResourceTagList, you must have console operator rights.

## NCP Calls

0x2222 123 15 Get NLM's Resource Tag List

## See Also

NWGetNLMInfo (page 122), NWGetNLMLoadedList (page 124)

# NWGetOSVersionInfo

Returns the NetWare OS version information.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetOSVersionInfo  (
   NWCONN_HANDLE                    conn,
   NWFSE_OS_VERSION_INFO  N_FAR   *fseOSVersionInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetOSVersionInfo
  (conn : NWCONN_HANDLE;
   Var fseOSVersionInfo : NWFSE_OS_VERSION_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fseOSVersionInfo**

(OUT) Points to the NWFSE_OS_VERSION_INFO structure, which contains the operating system version information.

## Return Values

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

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

## Remarks

Console operator rights and an authenticated connection are NOT necessary to call NWGetOSVersionInfo; you only need an attachment.

It is recommended that the newer function call, NWGetNetWareProductVersion (page 114), be used rather than NWGetOSVersionInfo.

## NCP Calls

0x2222 123 13 Get Operating System Version Information

# NWGetPacketBurstInfo

Returns the server's packet burst operational counters and statistics.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetPacketBurstInfo  (
   NWCONN_HANDLE                      conn,
   NWFSE_PACKET_BURST_INFO  N_FAR   *fsePacketBurstInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetPacketBurstInfo
  (conn : NWCONN_HANDLE;
   Var fsePacketBurstInfo : NWFSE_PACKET_BURST_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**fsePacketBurstInfo**

(OUT) Points to NWFSE_PACKET_BURST_INFO getting packet burst information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetPacketBurstInfo, you must have console operator rights.

## NCP Calls

0x2222 123 05 Packet Burst Information

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

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

# NWGetProtocolStackConfigInfo

Returns configuration information about the protocols on a 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetProtocolStackConfigInfo  (
   NWCONN_HANDLE                          conn,
   nuint32                                stackNum,
   pnstr8                                 stackFullName,
   NWFSE_PROTOCOL_STK_CONFIG_INFO  N_FAR  *fseProtocolStkConfigInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetProtocolStackConfigInfo
  (conn : NWCONN_HANDLE;
   stackNum : nuint32;
   stackFullName : pnstr8;
   Var fseProtocolStkConfigInfo : NWFSE_PROTOCOL_STK_CONFIG_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**stackNum**

(IN) Specifies the number of the protocol stack to return information about, usually returned by NWGetActiveProtocolStacks,

**stackFullName**

(OUT) Points to the full description of the protocol stack.

**fseProtocolStkConfigInfo**

(OUT) Points to NWFSE_PROTOCOL_STK_CONFIG_INFO getting protocol stack configuration information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89FF | Failure |

## Remarks

To call NWGetProtocolStackConfigInfo, you must have console operator rights.

## NCP Calls

0x2222 123 41 Get Protocol Stack Configuration Information

## See Also

NWGetActiveProtocolStacks (page 49), NWGetProtocolStackCustomInfo (page 135), NWGetProtocolStackStatsInfo (page 137), NWGetProtocolStkNumsByLANBrdNum (page 139), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetProtocolStackCustomInfo

Returns custom information about a protocol stack on a 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetProtocolStackCustomInfo  (
   NWCONN_HANDLE                          conn,
   nuint32                                stackNum,
   nuint32                                customStartNum,
   NWFSE_PROTOCOL_CUSTOM_INFO  N_FAR   *fseProtocolStackCustomInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetProtocolStackCustomInfo
  (conn : NWCONN_HANDLE;
   stackNum : nuint32;
   customStartNum : nuint32;
   Var fseProtocolStackCustomInfo : NWFSE_PROTOCOL_CUSTOM_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**stackNum**

(IN) Specifies the number identifying the protocol stack to return information about, usually returned by NWGetActiveProtocolStacks.

**customStartNum**

(IN) Specifies the custom information to begin with. Normally zero (0) on the first call; on all subsequent call, the previous `customStartNum`, plus the value returned in `customCount` of NWFSE_PROTOCOL_CUSTOM_INFO.

**fseProtocolStackCustomInfo**

(OUT) Points to NWFSE_PROTOCOL_CUSTOM_INFO getting protocol stack custom information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89FF | Failure |

## Remarks

To call NWGetProtocolStackCustomInfo, you must have console operator rights.

## NCP Calls

0x2222 123 43 Get Protocol Stack Custom Information

## See Also

NWGetActiveProtocolStacks (page 49), NWGetProtocolStackConfigInfo (page 133), NWGetProtocolStackStatsInfo (page 137), NWGetProtocolStkNumsByLANBrdNum (page 139), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetProtocolStackStatsInfo

Returns the protocol statistics indicated by `stackNum` for a 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetProtocolStackStatsInfo  (
   NWCONN_HANDLE                            conn,
   nuint32                                  stackNum,
   NWFSE_PROTOCOL_STK_STATS_INFO  N_FAR   *fseProtocolStkStatsInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetProtocolStackStatsInfo
  (conn : NWCONN_HANDLE;
   stackNum : nuint32;
   Var fseProtocolStkStatsInfo : NWFSE_PROTOCOL_STK_STATS_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**stackNum**

(IN) Specifies the number identifying the protocol stack to return information about, usually returned by NWGetActiveProtocolStacks.

**fseProtocolStkStatsInfo**

(OUT) Points to NWFSE_PROTOCOL_STK_STATS_INFO getting protocol stack configuration information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FF | Failure |

## Remarks

To call NWGetProtocolStackStatsInfo, you must have console operator rights.

## NCP Calls

0x2222 123 42 Get Protocol Stack Statistics Information

## See Also

NWGetActiveProtocolStacks (page 49), NWGetProtocolStackConfigInfo (page 133), NWGetProtocolStackCustomInfo (page 135), NWGetProtocolStkNumsByLANBrdNum (page 139), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetProtocolStkNumsByLANBrdNum

Returns a list of protocol stack ID numbers for a given LAN board.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetProtocolStkNumsByLANBrdNum  (
   NWCONN_HANDLE                    conn,
   nuint32                          LANBoardNum,
   NWFSE_PROTOCOL_ID_NUMS  N_FAR   *fseProtocolStkIDNums);
```

## Delphi Syntax

```
uses calwin32

Function NWGetProtocolStkNumsByLANBrdNum
  (conn : NWCONN_HANDLE;
   LANBoardNum : nuint32;
   Var fseProtocolStkIDNums : NWFSE_PROTOCOL_ID_NUMS
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**LANBoardNum**

(IN) Specifies the ID number of the LAN board for which you want a list of protocols; normally an ID number returned by NWGetActiveLANBoardList.

**fseProtocolStkIDNums**

(OUT) Points to NWFSE_PROTOCOL_ID_NUMS getting protocol stack numbers by LAN board number.

## 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 |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FF | Failure |

## Remarks

To call NWGetProtocolStkNumsByLANBrdNum, you must have console operator rights.

## NCP Calls

0x2222 123 45 Get Protocol Stack Numbers By LAN Board Number

## See Also

NWGetActiveLANBoardList (page 47), NWGetActiveProtocolStacks (page 49), NWGetProtocolStackConfigInfo (page 133), NWGetProtocolStackCustomInfo (page 135), NWGetProtocolStackStatsInfo (page 137), NWGetProtocolStkNumsByMediaNum (page 141)

# NWGetProtocolStkNumsByMediaNum

Returns a list of protocol stack ID numbers for a given media.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetProtocolStkNumsByMediaNum  (
   NWCONN_HANDLE                    conn,
   nuint32                          mediaNum,
   NWFSE_PROTOCOL_ID_NUMS  N_FAR   *fseProtocolStkIDNums);
```

## Delphi Syntax

```
uses calwin32

Function NWGetProtocolStkNumsByMediaNum
  (conn : NWCONN_HANDLE;
   mediaNum : nuint32;
   Var fseProtocolStkIDNums : NWFSE_PROTOCOL_ID_NUMS
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**mediaNum**

(IN) Specifies the ID number representing the frame type used by the protocol.

**fseProtocolStkIDNums**

(OUT) Points to NWFSE_PROTOCOL_ID_NUMS getting protocol stack numbers by media number.

## 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 |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

An invalid media number can abend the server

## Remarks

To call NWGetProtocolStkNumsByMediaNum, you must have console operator rights.

## NCP Calls

0x2222 123 44 Get Protocol Stack Number By Media Number

## See Also

NWGetActiveProtocolStacks (page 49), NWGetLoadedMediaNumList (page 95), NWGetProtocolStackConfigInfo (page 133), NWGetProtocolStackCustomInfo (page 135), NWGetProtocolStackStatsInfo (page 137), NWGetProtocolStkNumsByLANBrdNum (page 139)

# NWGetServerConnInfo

Returns connection information in a NetWare server for a given connection number.

**Local servers:** blocking

**Remote servers:**

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

**Platform:** NLM, Windows 95, Windows NT

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

**Service:** Server Environment

## Syntax

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

int NWGetServerConnInfo (
   NWCONN_HANDLE                        conn,
   nuint32                              retInfoMask,
   nuint32                              connectionNumber,
   SERVER_AND_VCONSOLE_INFO N_FAR      *serverTimeAndVConsoleInfo,
   puint16                              reserved,
   NWFSE_NETWORK_ADDRESS N_FAR         *networkAddress,
   NWFSE_LOGIN_TIME N_FAR              *loginTime,
   NWFSE_LOGIN_NAME N_FAR              *loginName,
   NWFSE_LOCK_INFO N_FAR               *lockInfo,
   NWFSE_PRINT_INFO N_FAR              *printInfo,
   NWFSE_STATS_INFO N_FAR              *statsInfo,
   NWFSE_ACCT_INFO N_FAR               *acctInfo,
   NWFSE_AUTH_INFO N_FAR               *authInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetServerConnInfo
  (conn : NWCONN_HANDLE;
   retInfoMask : nuint32;
   connectionNumber : nuint32;
   Var serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
   reserved : pnuint16;
   Var networkAddress : NWFSE_NETWORK_ADDRESS;
   Var loginTime : NWFSE_LOGIN_TIME;
   Var loginName : NWFSE_LOGIN_NAME;
   Var lockinfo : NWFSE_LOCK_INFO;
   Var printInfo : NWFSE_PRINT_INFO;
   Var statusInfo : NWFSE_STATS_INFO;
   Var acctInfo : NWFSE_ACCT_INFO;
```

```
   Var authInfo : NWFSE_AUTH_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**retInfoMask**

(IN) Specifies the connection information to be returned.

**connectionNumber**

(IN) Specifies the number of a connection to return information for.

**serverTimeAndVConsoleInfo**

(OUT) Points to SERVER_AND_VCONSOLE_INFO, which contains the console and server versions.

**reserved**

Is reserved for future use.

**networkAddress**

(OUT) Points to NWFSE_NETWORK_ADDRESS, which contains information about the connection address.

**loginTime**

(OUT) Points to NWFSE_LOGIN_TIME, which contains the server login time.

**loginName**

(OUT) Points to NWFSE_LOGIN_NAME, which contains the login name for the server.

**lockInfo**

(OUT) Points to NWFSE_LOCK_INFO, which contains information about the logical and record locks of the server.

**printInfo**

(OUT) Points to NWFSE_PRINT_INFO, which contains the printing information.

**statsInfo**

(OUT) Points to NWFSE_STATS_INFO, which contains statistical information about the server.

**acctInfo**

(OUT) Points to NWFSE_ACCT_INFO, which contains accounting information about the server.

**authInfo**

(OUT) Points to NWFSE_AUTH_INFO, which contains authentication information about the server.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | INVALID_CONNECTION |
| 0x880E | NWE_BUFFER_OVERFLOW |
| 0x8977 | NEW_BUFFER_TOO_SMALL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

Before calling NWGetServerConnInfo, allocate memory for the address pointer used in NWFSE_NETWORK_ADDRESS and set the `addressSize` field to a valid value.

If the passed in `addressSize` value is smaller than expected, `addressSize` is set to the required value and NWGetServerConnInfo returns NWE_BUFFER_OVERFLOW.

**WARNING:** If memory isn't allocated and invalid information is passed to `networkAddress`, the server might abend.

`retInfoMask` can have the following values (all can be ORed with the exception of CONN_INFO_ALL_MASK):

| Value | Delphi | Definition |
|---|---|---|
| 0x00000001 | $00000001 | CONN_INFO_TRANS_MASK specifies to return NWFSE_NETWORK_ADDRESS (page 264). |
| 0x00000002 | $00000002 | CONN_INFO_LOGIN_TIME_MASK specifies to return NWFSE_LOGIN_TIME (page 255). |
| 0x00000004 | $00000004 | CONN_INFO_LOGIN_NAME_MASK specifies to return NWFSE_LOGIN_NAME (page 254). |
| 0x00000008 | $00000008 | CONN_INFO_LOCK_MASK specifies to return NWFSE_LOCK_INFO (page 253). |
| 0x00000010 | $00000010 | CONN_INFO_PRINT_MASK specifies to return NWFSE_PRINT_INFO (page 276). |
| 0x00000020 | $00000020 | CONN_INFO_STATS_MASK specifies to return NWFSE_STATS_INFO (page 289). |
| 0x00000040 | $00000040 | CONN_INFO_ACCT_MASK specifies to return NWFSE_ACCT_INFO (page 226). |
| 0x00000080 | $00000080 | CONN_INFO_AUTH_MASK specifies to return NWFSE_AUTH_INFO (page 233). |
| 0xFFFFFFFF | | CONN_INFO_ALL_MASK specifies that all structures are to be returned. |

## NCP Calls

0x2222 123 16 Enumerate Connection Information from Connection List

# NWGetServerInfo

Returns the address and the number of hops to the server specified by `serverName` in relation to the server represented by `conn` .

**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:** Server Environment

## Syntax

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

NWCCODE N_API NWGetServerInfo (
   NWCONN_HANDLE              conn,
   nuint32                    serverType,
   const nstr8 N_FAR         *serverName,
   NWFSE_SERVER_INFO N_FAR   *fseServerInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetServerInfo
  (conn : NWCONN_HANDLE;
   serverType : nuint32;
   const serverName : pnstr8;
   Var fseServerInfo : NWFSE_SERVER_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**serverType**

(IN) Specifies the type of server to search for, such as a file server (0x0400).

**serverName**

(IN) Points to the name of the server for which to search.

**fseServerInfo**

(OUT) Points to NWFSE_SERVER_INFO containing server information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | Invalid Server Name |
| 0x8901 | Invalid Server Type |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |
| 0x89FB | INVALID_PARAMETERS |

## Remarks

To call NWGetServerInfo, you must have console operator rights to the 4.x server indicated by `conn`. `serverType` can indicate either a 3.x or 4.x server.

## NCP Calls

0x2222 123 54 Get Server Information

## See Also

NWGetServerSourcesInfo (page 153)

# NWGetServerSetCategories

Returns SET console command configuration parameter categories for the 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetServerSetCategories  (
   NWCONN_HANDLE                         conn,
   nuint32                               startNum,
   NWFSE_SERVER_SET_CATEGORIES  N_FAR   *fseServerSetCategories);
```

## Delphi Syntax

```
uses calwin32

Function NWGetServerSetCategories
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   Var fseServerSetCategories : NWFSE_SERVER_SET_CATEGORIES
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the value returned in `nextSequenceNumber` of `fseServerSetCategories`; normally zero (0) on the first call.

**fseServerSetCategories**

(OUT) Points to NWFSE_SERVER_SET_CATEGORIES getting server set categories.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89F5 | Invalid Star Number |

## Remarks

To call NWGetServerSetCategories, you must have console operator rights.

## NCP Calls

0x2222 123 61 Get Server Set Categories

## See Also

NWGetServerSetCommandsInfo (page 151), NWSMSetDynamicCmdIntValue (page 349), NWSMSetDynamicCmdStrValue (page 351)

# NWGetServerSetCommandsInfo

Returns SET console command configuration parameter commands for the 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetServerSetCommandsInfo  (
   NWCONN_HANDLE                          conn,
   nuint32                                startNum,
   NWFSE_SERVER_SET_CMDS_INFO  N_FAR   *fseServerSetCmdsInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetServerSetCommandsInfo
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   Var fseServerSetCmdsInfo : NWFSE_SERVER_SET_CMDS_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the value returned in `nextSequenceNumber` of `fseServerSetCmdsInfo` ; normally zero (0) on the first call.

**fseServerSetCmdsInfo**

(OUT) Points to NWFSE_SERVER_SET_CMDS_INFO getting server set commands information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89FF | Failure of Invalid Start Number |

## Remarks

To call NWGetServerSetCommandsInfo, you must have console operator rights.

## NCP Calls

0x2222 123 60 Get Server Set Commands Information

## See Also

NWGetServerSetCategories (page 149), NWSMSetDynamicCmdIntValue (page 349), NWSMSetDynamicCmdStrValue (page 351)

# NWGetServerSourcesInfo

Returns address information about servers known to a server with a given name.

**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:** Server Environment

## Syntax

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

NWCCODE N_API NWGetServerSourcesInfo (
   NWCONN_HANDLE                  conn,
   nuint32                        startNum,
   nuint32                        serverType,
   const nstr8 N_FAR             *serverName,
   NWFSE_SERVER_SRC_INFO N_FAR   *fseServerSrcInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetServerSourcesInfo
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   serverType : nuint32;
   const serverName : pnstr8;
   Var fseServerSrcInfo : NWFSE_SERVER_SRC_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the value returned in `numberOfEntries` of `fseServerSrcInfo`; normally zero (0) on the first call.

**serverType**

(IN) Specifies the server type to get information from, such as a file server (0x0400).

**serverName**

(IN) Points to the server name to get information from.

**fseServerSetCmdsInfo**

(OUT) Points to NWFSE_SERVER_SRC_INFO getting server sources information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8801 | Invalid Server Name or Server Type |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |

## Remarks

To call NWGetServerSourcesInfo, you must have console operator rights.

## NCP Calls

0x2222 123 55 Get Server Sources Information

## See Also

NWGetServerInfo (page 147)

# NWGetUserInfo

Gets NetWare user information about a given logged-in server connection.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetUserInfo  (
   NWCONN_HANDLE             conn,
   nuint32                   connNum,
   pnstr8                    userName,
   NWFSE_USER_INFO  N_FAR   *fseUserInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetUserInfo
  (conn : NWCONN_HANDLE;
   connNum : nuint32;
   userName : pnstr8;
   Var fseUserInfo : NWFSE_USER_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**connNum**

(IN) Specifies the connection number of logged-in user.

**userName**

(OUT) Points to the user name (size of MAX_DN_BYTES).

**fseServerSetCmdsInfo**

(OUT) Points to NWFSE_USER_INFO getting user information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89FF | Failure or Invalid Connection Number |

## Remarks

To call NWGetUserInfo, you must have console operator rights.

## NCP Calls

0x2222 123 04 User Information

## See Also

NWCCGetAllConnInfo, NWGetActiveConnListByType (page 45), NWGetConnectionInformation

# NWGetVolumeInfoByLevel

Returns information about the specified volume, returning different structures depending on the `infoLevel` specified.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetVolumeInfoByLevel  (
   NWCONN_HANDLE                        conn,
   nuint32                              volNum,
   nuint32                              infoLevel,
   NWFSE_VOLUME_INFO_BY_LEVEL  N_FAR   *fseVolumeInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetVolumeInfoByLevel
  (conn : NWCONN_HANDLE;
   volNum : nuint32;
   infoLevel : nuint32;
   Var fseVolumeInfo : NWFSE_VOLUME_INFO_BY_LEVEL
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**volNum**

(IN) Specifies the volume number for which information is being obtained.

**infoLevel**

(IN) Specifies which level of information to return (1 or 2).

**fseVolumeInfo**

(OUT) Points to NWFSE_VOLUME_INFO_BY_LEVEL containing the information.

## 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 |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x8998 | VOLUME_DOES_NOT_EXIST |
| 0x89FF | Failure |

## Remarks

Console operator rights are NOT necessary to call NWGetVolumeInfoByLevel.

## NCP Calls

0x2222 123 34 Get Volume Information By Level

## See Also

NWGetVolumeInfoWithHandle, NWGetVolumeInfoWithNumber (Volume Management)

# NWGetVolumeSegmentList

Returns a list of up to 32 volume segments for a given volume.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetVolumeSegmentList  (
   NWCONN_HANDLE                        conn,
   nuint32                              volumeNum,
   NWFSE_VOLUME_SEGMENT_LIST  N_FAR   *fseVolumeSegmentList);
```

## Delphi Syntax

```
uses calwin32

Function NWGetVolumeSegmentList
  (conn : NWCONN_HANDLE;
   volNum : nuint32;
   Var fseVolumeSegmentList : NWFSE_VOLUME_SEGMENT_LIST
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**volumeNum**

(IN) Specifies the number representing a specific volume. Zero (0) = Volume SYS

**fseVolumeSegmentList**

(OUT) Points to NWFSE_VOLUME_SEGMENT_LIST containing the volume segment list.

## 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 |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x8998 | VOLUME_DOES_NOT_EXIST |

## Remarks

Console operator rights are NOT necessary to call NWGetVolumeSegmentList.

## NCP Calls

0x2222 123 33 Get Volume Segment List

# NWGetVolumeSwitchInfo

Gets information about the volume switch counter such as the number of overall times through the function.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWGetVolumeSwitchInfo  (
   NWCONN_HANDLE                      conn,
   nuint32                            startNum,
   NWFSE_VOLUME_SWITCH_INFO  N_FAR   *fseVolumeSwitchInfo);
```

## Delphi Syntax

```
uses calwin32

Function NWGetVolumeSwitchInfo
  (conn : NWCONN_HANDLE;
   startNum : nuint32;
   Var fseVolumeSwitchInfo : NWFSE_VOLUME_SWITCH_INFO
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**startNum**

(IN) Specifies the starting number; set to zero (0) on the first call.

**fseVolumeSwitchInfo**

(OUT) Points to NWFSE_VOLUME_SWITCH_INFO getting volume switch information.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x897E | NCP_BOUNDARY_CHECK_FAILED |
| 0x89FF | Failure or Invalid Start Item Number |

## Remarks

To call NWGetVolumeSwitchInfo, you must have console operator rights.

## NCP Calls

0x2222 123 09 Volume Switch Information

# NWIsManager

Checks whether a calling station is a manager.

**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:** Server Environment

## Syntax

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

NWCCODE N_API  NWIsManager  (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

Function NWIsManager
  (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 | Calling station is a manager. |
| 0x89FF | Calling station is not a manager. |

## Remarks

To call NWIsManager, you must have console operator rights.

A station is a manager if it is a supervisor, or if it appears in the MANAGERS property of the supervisor object.

## NCP Calls

0x2222 23 73 Is Calling Station A Manager

# NWLoginToFileServer

Attempts to log an object in to 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:** Server Environment

## Syntax

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

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

## Delphi Syntax

```
uses calwin32

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

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**objName**

(IN) Points to the bindery object name of the object logging in to the NetWare server object name (up to 48 characters including the NULL terminator).

**objType**

(IN) Specifies the Bindery object type of the object logging in to the NetWare server.

**password**

(IN) Points to the upper-case password of the bindery object logging in to the NetWare server (or pass in a zero-length string if a password is not required).

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESSFUL |
| 0x8800 | ALREADY_ATTACHED |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x8996 | SERVER_OUT_OF_MEMORY |
| 0x89C1 | LOGIN_DENIED_NO_ACCOUNT_BALANCE |
| 0x89C2 | LOGIN_DENIED_NO_CREDIT |
| 0x89C5 | INTRUDER_DETECTION_LOCK |
| 0x89D9 | ERR_MAX_SERVERS |
| 0x89DA | UNAUTHORIZED_LOGIN_TIME |
| 0x89DB | UNAUTHORIZED_LOGIN_STATION |
| 0x89DC | ACCOUNT_DISABLED |
| 0x89DE | PASSWORD_HAS_EXPIRED_NO_GRACE |
| 0x89DF | PASSWORD_EXPIRED |
| 0x89FB | INVALID_PARAMETERS |
| 0x89FE | BINDERY_LOCKED |
| 0x89FF | NO_SUCH_OBJECT_OR_BAD_PASSWORD |
| 0xFF | Failure |

## Remarks

If the `conn` parameter specifies an NDS authenticated connection, NWLoginToFileServer will return ALREADY_ATTACHED (0x8800).

To determine if the `conn` parameter specifies an NDS authenticated connection, call NWCCGetConnRefInfo with the `infoType` parameter set to NWCC_INFO_NDS_STATE. If NWCC_NDS_CAPABLE is returned in the `infoType` parameter, the connection must be cleared (see Closing and Clearing Connections) and a new connection must be opened (see Attaching to Servers and Opening Connections) before NWLoginToFileServer will successfully return (*NDK: Connection, Message, and NCP Extensions*).

If the encryption key is not available, it attempts an unencrypted login. If the key is available, the password is encrypted and an encrypted login is attempted.

NWLoginToFileServer performs only the login, not the attach. Clients must be previously attached to call NWLoginToFileServer. Attaching to a NetWare server is not the same as logging in. A workstation attaches to a NetWare server to obtain a connection number. The workstation can then log in to the NetWare server using that connection number. NWLoginToFileServer does not, however, interpret the login script.

Valid bindery object types for OT_ constants follow:

```
OT_WILD                    0xFFFF
OT_UNKNOWN                 0x0000
OT_USER                    0x0100
OT_USER_GROUP              0x0200
OT_PRINT_QUEUE             0x0300
OT_FILE_SERVER             0x0400
OT_JOB_SERVER              0x0500
OT_GATEWAY                 0x0600
OT_PRINT_SERVER            0x0700
OT_ARCHIVE_QUEUE           0x0800
OT_ARCHIVE_SERVER          0x0900
OT_JOB_QUEUE               0x0A00
OT_ADMINISTRATION          0x0B00
OT_NAS_SNA_GATEWAY         0x2100
OT_REMOTE_BRIDGE_SERVER    0x2600
OT_TCPIP_GATEWAY           0x2700
```

Extended bindery object types follow:

```
OT_TIME_SYNCHRONIZATION_SERVER   0x2D00
OT_ARCHIVE_SERVER_DYNAMIC_SAP    0x2E00
OT_ADVERTISING_PRINT_SERVER      0x4700
OT_PRINT_QUEUE_USER              0x5300
```

## NCP Calls

0x2222 23 24 Keyed Object Login
0x2222 23 53 Get Bindery Object ID

# NWLogoutFromFileServer

Attempts to log the workstation out of the specified NetWare 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWLogoutFromFileServer  (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

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

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle associated with the server from which to log out.

## 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 |
| 0x88FB | INVALID_PARAMETERS |

## Remarks

To call NWLogoutFromFileServer, you must have console operator rights.

If successful, drive mappings dependent on the connection are deleted.

# NWSetFileServerDateAndTime

Sets the date and time of a 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:** Server Environment

## Syntax

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

NWCCODE N_API  NWSetFileServerDateAndTime  (
   NWCONN_HANDLE   conn,
   nuint8          year,
   nuint8          month,
   nuint8          day,
   nuint8          hour,
   nuint8          minute,
   nuint8          second);
```

## Delphi Syntax

```
uses calwin32

Function NWSetFileServerDateAndTime
  (conn : NWCONN_HANDLE;
   year : nuint8;
   month : nuint8;
   day : nuint8;
   hour : nuint8;
   minute : nuint8;
   second : nuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**year**

(IN) Specifies the value corresponding to the year (0-179).

**month**

(IN) Specifies the month value (1=January; 12=December).

**day**

(IN) Specifies the day value (1-31).

**hour**

(IN) Specifies the hour value (0=midnight; 23=11 p.m.).

**minute**

(IN) Specifies the minute value (0-59).

**second**

(IN) Specifies the second value (0-59).

## 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 |

## Remarks

To call NWSetFileServerDateAndTime, you must have console operator rights.

The `year` parameter contains the following values which correspond to the specified years:

0-79 2000-2079
80-99 1980-1999
100-179 2000-2079

## NCP Calls

0x2222 23 202 Set File Server Date And Time

# Server Environment Structures

# 3

This documentation alphabetically lists the reference information for each server environment structure.

# CACHE_COUNTERS

Returns cache information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   nuint32       readExistingBlockCount ;
   nuint32       readExistingWriteWaitCount ;
   nuint32       readExistingPartialReadCount ;
   nuint32       readExistingReadErrorCount ;
   nuint32       writeBlockCount ;
   nuint32       writeEntireBlockCount ;
   nuint32       getDiskCount ;
   nuint32       getDiskNeedToAllocCount ;
   nuint32       getDiskSomeoneBeatMeCount ;
   nuint32       getDiskPartialReadCount ;
   nuint32       getDiskReadErrorCount ;
   nuint32       getAsyncDiskCount ;
   nuint32       getAsyncDiskNeedToAlloc ;
   nuint32       getAsyncDiskSomeoneBeatMe ;
   nuint32       errorDoingAsyncReadCount ;
   nuint32       getDiskNoReadCount ;
   nuint32       getDiskNoReadAllocCount ;
   nuint32       getDiskNoReadSomeoneBeatMeCount ;
   nuint32       diskWriteCount ;
   nuint32       diskWriteAllocCount ;
   nuint32       diskWriteSomeoneBeatMeCount ;
   nuint32       writeErrorCount ;
   nuint32       waitOnSemaphoreCount ;
   nuint32       allocBlockWaitForSomeoneCount ;
   nuint32       allocBlockCount ;
   nuint32       allocBlockWaitCount ;
} CACHE_COUNTERS;
```

## Delphi Structure

```
uses calwin32

  CACHE_COUNTERS = packed Record
    readExistingBlockCount : nuint32;
    readExistingWriteWaitCount : nuint32;
    readExistingPartialReadCount : nuint32;
    readExistingReadErrorCount : nuint32;
    writeBlockCount : nuint32;
    writeEntireBlockCount : nuint32;
    getDiskCount : nuint32;
    getDiskNeedToAllocCount : nuint32;
    getDiskSomeoneBeatMeCount : nuint32;
```

```
    getDiskPartialReadCount : nuint32;
    getDiskReadErrorCount : nuint32;
    getAsyncDiskCount : nuint32;
    getAsyncDiskNeedToAlloc : nuint32;
    getAsyncDiskSomeoneBeatMe : nuint32;
    errorDoingAsyncReadCount : nuint32;
    getDiskNoReadCount : nuint32;
    getDiskNoReadAllocCount : nuint32;
    getDiskNoReadSomeoneBeatMeCount : nuint32;
    diskWriteCount : nuint32;
    diskWriteAllocCount : nuint32;
    diskWriteSomeoneBeatMeCount : nuint32;
    writeErrorCount : nuint32;
    waitOnSemaphoreCount : nuint32;
    allocBlockWaitForSomeoneCount : nuint32;
    allocBlockCount : nuint32;
    allocBlockWaitCount : nuint32
  End;
```

## Fields

**readExistingBlockCount**

Specifies the number of times an existing cache block has been read.

**readExistingWriteWaitCount**

Specifies the number of times an existing cache block was being read but had to wait until someone else was finished writing to it before it could be read completely.

**readExistingPartialReadCount**

Specifies the number cache blocks that were being read but encountered dirty sectors.

**readExistingReadErrorCount**

Specifies the number of cache blocks that experienced errors while trying to be read.

**writeBlockCount**

Specifies the number of cache buffers that were dirty and written to disk.

**writeEntireBlockCount**

Specifies the number of entire cache blocks that were dirty and written to disk.

**getDiskCount**

Specifies the number of times a block is obtained from disk to be compared against what is in a cache.

**getDiskNeedToAllocCount**

Specifies the number of times a cache control structure needs to be allocated because what is obtained from disk is not in a cache.

**getDiskSomeoneBeatMeCount**

Specifies the number of times a cache control structure is allocated to store new data from a disk but a recheck of a cache shows the new data is already stored indicating someone else beat

the first attempt to store the data. (The newly allocated cache control structure is then returned to the available list.)

**getDiskPartialReadCount**

Specifies the number of times a disk was only partially read.

**getDiskReadErrorCount**

Specifies the number of times an error was encountered while reading data from a disk.

**getAsyncDiskCount**

Specifies the number of times a block is obtained from an asynchronous disk to be compared against what is in a cache.

**getAsyncDiskNeedToAlloc**

Specifies the number of times a cache control structure needs to be allocated because what is obtained from an asynchronous disk is not in a cache.

**getAsyncDiskSomeoneBeatMe**

Specifies the number of times a cache control structure is allocated to store new data from an asynchronous disk but a recheck of a cache shows the new data is already stored indicating someone else beat the first attempt to store the data. (The newly allocated cache control structure is then returned to the available list.)

**errorDoingAsyncReadCount**

Specifies the number of times an error occurred while reading from a disk. It is used to compare with what may or may not be in a cache.

**getDiskNoReadCount**

Specifies the number of times data is obtained from a disk and put into a cache without requesting a read on the data.

**getDiskNoReadAllocCount**

Specifies the number of times a new cache control structure has to be allocated during the time that data is obtained from disk to be put into a cache.

**getDiskNoReadSomeoneBeatMeCount**

Specifies the number of times a cache control structure has to be allocated during the time that data is obtained from disk to be put into a cache but a recheck of a cache shows the new data is already stored indicating someone else beat the first attempt to store the data. (The newly allocated cache control structure is then returned to the available list.)

**diskWriteCount**

Specifies the number of times a cache block has been written to disk.

**diskWriteAllocCount**

Specifies the number of times a cache control structure was allocated during the time that data is being written to disk.

**diskWriteSomeoneBeatMeCount**

Specifies the number of times a cache control structure has to be allocated during the time that a cache block is written to disk but a recheck of a cache shows the new data is already written

to disk indicating someone else beat the first attempt to store the data. (The newly allocated cache control structure is then returned to the available list.)

**writeErrorCount**

Specifies the number of times an error was encountered while writing a cache to disk.

**waitOnSemaphoreCount**

Specifies the number of times a cache control blocks was waiting on a semaphore while cache and disk blocks were being checked

**allocBlockWaitForSomeoneCount**

Specifies the number of times that the allocate waiting count was set. You must set a semaphore and try again later.

**allocBlockCount**

Specifies the number of times a cache control block was allocated.

**allocBlockWaitCount**

Specifies the number of times the LRU and cache nodes were not available.

# CACHE_INFO

Returns information about a cache.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   nuint32   maxByteCount ;
   nuint32   minNumOfCacheBuffers ;
   nuint32   minCacheReportThreshold ;
   nuint32   allocWaitingCount ;
   nuint32   numDirtyBlocks ;
   nuint32   cacheDirtyWaitTime ;
   nuint32   cacheMaxConcurrentWrites ;
   nuint32   maxDirtyTime ;
   nuint32   numOfDirCacheBuffers ;
   nuint32   cacheByteToBlockShiftFactor ;
} CACHE_INFO;
```

## Delphi Structure

```
uses calwin32

  CACHE_INFO = packed Record
    maxByteCount : nuint32;
    minNumOfCacheBuffers : nuint32;
    minCacheReportThreshold : nuint32;
    allocWaitingCount : nuint32;
    numDirtyBlocks : nuint32;
    cacheDirtyWaitTime : nuint32;
    cacheMaxConcurrentWrites : nuint32;
    maxDirtyTime : nuint32;
    numOfDirCacheBuffers : nuint32;
    cacheByteToBlockShiftFactor : nuint32
  End;
```

## Fields

**maxByteCount**

Specifies the length in bytes of a cache block.

**minNumOfCacheBuffers**

Specifies the minimum number of cache buffers allowed on the server (default is 20, but values from 20-1000 are supported).

**minCacheReportThreshold**

Specifies the number of cache buffers used for the report threshold (default value is 20, but values from 0-1000 are supported).

**allocWaitingCount**

Specifies the number of processes waiting to allocate a cache block.

**numDirtyBlocks**

Specifies the number of dirty blocks waiting to be written to disk.

**cacheDirtyWaitTime**

Specifies the maximum wait before a Write request is written to disk (default is 3.3 seconds, but values from 0.1-10 seconds are supported).

**cacheMaxConcurrentWrites**

Specifies the maximum number of Write requests for changed file data that can be put in the elevator before the disk head begins a sweep across the disk (default is 50, but values from 10-100 are supported).

**maxDirtyTime**

Specifies the longest time (in ticks) since the server was brought up that a dirty block has waited before it was written to disk.

**numOfDirCacheBuffers**

Specifies the number of directory cache buffers on the server.

**cacheByteToBlockShiftFactor**

Specifies the `n` factor used in the block size equation.

## Remarks

The `minNumOfCacheBuffers`, `minCacheReportThreshold`, `cacheDirtyWaitTime`, and `cacheMaxConcurrentWrites` fields can be set by using the SET console command.

When the number of cache buffers reach a number equal to the sum of the numbers specified by the `minNumOfCacheBuffers` and `minCacheReportThreshold` fields, the server sends a message warning that the cache buffers are getting low.

The block size (in bytes) is calculated using:

block size = $2^{n+9}$

where n is the shift factor.

# CACHE_MEM_COUNTERS

Returns cache memory information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   nuint32   originalNumOfCacheBuffers ;
   nuint32   currentNumOfCacheBuffers ;
   nuint32   cacheDirtyBlockThreshold ;
   nuint32   waitNodeCount ;
   nuint32   waitNodeAllocFailureCount ;
   nuint32   moveCacheNodeCount ;
   nuint32   moveCacheNodeFromAvailCount ;
   nuint32   accelerateCacheNodeWriteCount ;
   nuint32   removeCacheNodeCount ;
   nuint32   removeCacheNodeFromAvailCount ;
} CACHE_MEM_COUNTERS;
```

## Delphi Structure

```
uses calwin32

  CACHE_MEM_COUNTERS = packed Record
    originalNumOfCacheBuffers : nuint32;
    currentNumOfCacheBuffers : nuint32;
    cacheDirtyBlockThreshold : nuint32;
    waitNodeCount : nuint32;
    waitNodeAllocFailureCount : nuint32;
    moveCacheNodeCount : nuint32;
    moveCacheNodeFromAvailCount : nuint32;
    accelerateCacheNodeWriteCount : nuint32;
    removeCacheNodeCount : nuint32;
    removeCacheNodeFromAvailCount : nuint32
  End;
```

## Fields

**originalNumOfCacheBuffers**

Specifies the number of cache buffers that existed when the server was brought up.

**currentNumOfCacheBuffers**

Specifies the number of cache buffers currently on the server.

**cacheDirtyBlockThreshold**

Specifies the maximum number of cache blocks allowed to be dirty simultaneously.

**waitNodeCount**

Specifies the number of wait nodes that have been allocated. (Wait nodes are memory chunks created to track the start and end of internal processes.)

**waitNodeAllocFailureCount**

Specifies the number of times a wait node was unable to be allocated.

**moveCacheNodeCount**

Specifies the number of times a cache block control node has been moved from one node to another for memory management.

**moveCacheNodeFromAvailCount**

Specifies the number of times a cache block control node has been available and sitting in the available list and then moved.

**accelerateCacheNodeWriteCount**

Specifies the number of dirty cache nodes that were moved to the beginning of the list to be written to disk.

**removeCacheNodeCount**

Specifies the number of cache block control structure nodes that were removed while collapsing cache memory segments.

**removeCacheNodeFromAvailCount**

Specifies the number of cache block control nodes that were removed from the cache node available list.

# CACHE_TREND_COUNTERS

Returns cache trend information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   nuint32   numCacheChecks ;
   nuint32   numCacheHits ;
   nuint32   numDirtyCacheChecks ;
   nuint32   numDirtyCacheHits ;
   nuint32   cacheUsedWhileChecking ;
   nuint32   waitForDirtyBlocksDecreaseCount ;
   nuint32   allocBlockFromAvailCount ;
   nuint32   allocBlockFromLRUCount ;
   nuint32   allocBlockAlreadyWaiting ;
   nuint32   LRUSittingTime ;
} CACHE_TREND_COUNTERS;
```

## Delphi Structure

```
uses calwin32

  CACHE_TREND_COUNTERS = packed Record
     numCacheChecks : nuint32;
     numCacheHits : nuint32;
     numDirtyCacheChecks : nuint32;
     numDirtyCacheHits : nuint32;
     cacheUsedWhileChecking : nuint32;
     waitForDirtyBlocksDecreaseCount : nuint32;
     allocBlockFromAvailCount : nuint32;
     allocBlockFromLRUCount : nuint32;
     allocBlockAlreadyWaiting : nuint32;
     LRUSittingTime : nuint32
  End;
```

## Fields

**numCacheChecks**

Specifies the total number of times any block in the cache was looked at since the server was brought up.

**numCacheHits**

Specifies the number of times cache requests were serviced from existing cache blocks.

**numDirtyCacheChecks**

Specifies the number of times a cache block was checked to determine if it is dirty.

**numDirtyCacheHits**

Specifies the time in ticks the oldest cache block has been available (sitting in the LRU list).

**cacheUsedWhileChecking**

Specifies the number of cache blocks that were allocated and then returned to the available list while a cache was being checked during different cache operations (read, write, etc.).

**waitForDirtyBlocksDecreaseCount**

Specifies the number of times a process had to wait until the number of dirty cache blocks decreased to less than the cache dirty block threshold.

**allocBlockFromAvailCount**

Specifies the number of cache blocks removed from the available list and used.

**allocBlockFromLRUCount**

Specifies the number of cache blocks removed from the LRU list and used done if cache blocks cannot be removed from the available list).

**allocBlockAlreadyWaiting**

Specifies the number of times an attempt was made to allocate a cache block but none are available in the available list or LRU (system waits and tries again later).

**LRUSittingTime**

Specifies the time (in ticks) that the oldest cache block has been available and was sitting in the LRU list.

# CPU_INFO

Returns CPU information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   nuint32   pageTableOwnerFlag ;
   nuint32   CPUTypeFlag ;
   nuint32   coProcessorFlag ;
   nuint32   busTypeFlag ;
   nuint32   IOEngineFlag ;
   nuint32   FSEngineFlag ;
   nuint32   nonDedicatedFlag;
} CPU_INFO;
```

## Delphi Structure

```
uses calwin32

  CPU_INFO = packed Record
    pageTableOwnerFlag : nuint32;
    CPUTypeFlag : nuint32;
    coProcessorFlag : nuint32;
    busTypeFlag : nuint32;
    IOEngineFlag : nuint32;
    FSEngineFlag : nuint32;
    nonDedicatedFlag : nuint32;
  End;
```

## Fields

**pageTableOwnerFlag**

Specifies which domain is the current domain.

**CPUTypeFlag**

Specifies the CPU type:

0 80386
1 80486
2 Pentium
3 Pentium Pro

**coProcessorFlag**

Specifies whether a numeric coprocessor is present (true=present).

**busTypeFlag**

Specifies the bus type:

0x01=micro channel
0x02=EISA
0x04=PCI
0x08=PCMCIA
0x10=ISA

**`IOEngineFlag`**

Specifies whether the IO engine is installed (true=installed).

**`FSEngineFlag`**

Specifies whether the file system engine is installed (true=installed).

**`nonDedicatedFlag`**

Specifies whether the CPU is dedicated.

## Remarks

# DIR_CACHE_INFO

Returns information for a directory cache.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   nuint32   minTimeSinceFileDelete ;
   nuint32   absMinTimeSinceFileDelete ;
   nuint32   minNumOfDirCacheBuffers ;
   nuint32   maxNumOfDirCacheBuffers ;
   nuint32   numOfDirCacheBuffers ;
   nuint32   dCMinNonReferencedTime ;
   nuint32   dCWaitTimeBeforeNewBuffer ;
   nuint32   dCMaxConcurrentWrites ;
   nuint32   dCDirtyWaitTime ;
   nuint32   dCDoubleReadFlag ;
   nuint32   mapHashNodeCount ;
   nuint32   spaceRestrictionNodeCount ;
   nuint32   trusteeListNodeCount ;
   nuint32   percentOfVolumeUsedByDirs ;
} DIR_CACHE_INFO;
```

## Delphi Structure

```
uses calwin32

  DIR_CACHE_INFO = packed Record
    minTimeSinceFileDelete : nuint32;
    absMinTimeSinceFileDelete : nuint32;
    minNumOfDirCacheBuffers : nuint32;
    maxNumOfDirCacheBuffers : nuint32;
    numOfDirCacheBuffers : nuint32;
    dCMinNonReferencedTime : nuint32;
    dCWaitTimeBeforeNewBuffer : nuint32;
    dCMaxConcurrentWrites : nuint32;
    dCDirtyWaitTime : nuint32;
    dCDoubleReadFlag : nuint32;
    mapHashNodeCount : nuint32;
    spaceRestrictionNodeCount : nuint32;
    trusteeListNodeCount : nuint32;
    percentOfVolumeUsedByDirs : nuint32;
  End;
```

## Fields

**minTimeSinceFileDelete**

Specifies the minimum time (in clock ticks) between when a file is deleted and when it can be purged.

**absMinTimeSinceFileDelete**

Specifies the minimum time (in clock ticks) between when a file is deleted and when it can be purged after the system has no available blocks.

**minNumOfDirCacheBuffers**

Specifies the minimum number of directory cache buffers that can be allocated on the server.

**maxNumOfDirCacheBuffers**

Specifies the maximum number of directory cache buffers that can be allocated on the server.

**numOfDirCacheBuffers**

Specifies the current number of directory cache buffers on the server.

**dCMinNonReferencedTime**

Specifies the time (in clock ticks) that must elapse between the last reference of a directory buffer and the time it is reused.

**dCWaitTimeBeforeNewBuffer**

Specifies the time (in clock ticks) that must elapse before an additional directory cache buffer can be allocated.

**dCMaxConcurrentWrites**

Specifies the maximum number of write requests from directory cache buffers that can be put in the elevator before they are written to disk.

**dCDirtyWaitTime**

Specifies the maximum time (in clock ticks) that the server can wait before writing dirty cache buffers to disk.

**dCDoubleReadFlag**

Specifies whether the directory block must be read and verified from both copies of directory tables.

**mapHashNodeCount**

Specifies the number of times a hash node has been allocated for directories.

**spaceRestrictionNodeCount**

Specifies the total number of disk space restrictions placed since the server was brought up.

**trusteeListNodeCount**

Specifies the total number of trustee assignments set on the file system since the server was brought up.

**percentOfVolumeUsedByDirs**

Specifies the total volume space percentage that is used by directory entries.

## Remarks

The `minNumOfDirCacheBuffers`, `maxNumOfDirCacheBuffers`, `dCMinNonReferencedTime`, `dCWaitTimeBeforeNewBuffer`, `dCMaxConcurrentWrites`, `dCDirtyWaitTime`, and `percentOfVolumeUsedByDirs` fields can be set by using the SET console command.

# DRV_MAP_TABLE

Returns drive map table data.

**Service:** Server Environment

**Defined In:** nwserver.h

## Structure

```
typedef struct
{
   nuint32   systemElapsedTime ;
   nuint8    SFTSupportLevel ;
   nuint8    logicalDriveCount ;
   nuint8    physicalDriveCount ;
   nuint8    diskChannelTable [5];
   nuint16   pendingIOCommands ;
   nuint8    driveMappingTable [32];
   nuint8    driveMirrorTable [32];
   nuint8    deadMirrorTable [32];
   nuint8    reMirrorDriveNumber ;
   nuint8    reserved ;
   nuint32   reMirrorCurrentOffset ;
   nuint16   SFTErrorTable [60];
} DRV_MAP_TABLE;
```

## Delphi Structure

```
uses calwin32

  DRV_MAP_TABLE = packed Record
    systemElapsedTime : nuint32;
    SFTSupportLevel : nuint8;
    logicalDriveCount : nuint8;
    physicalDriveCount : nuint8;
    diskChannelTable : Array[0..4] Of nuint8;
    pendingIOCommands : nuint16;
    driveMappingTable : Array[0..31] Of nuint8;
    driveMirrorTable : Array[0..31] Of nuint8;
    deadMirrorTable : Array[0..31] Of nuint8;
    reMirrorDriveNumber : nuint8;
    reserved : nuint8;
    reMirrorCurrentOffset : nuint32;
    SFTErrorTable : Array[0..59] Of nuint16;
  End;
```

## Fields

**systemElapsedTime**

Specifies how long the NetWare server has been up. `systemElapsedTime` is returned in units of approximately 1/18 second and is used to determine the amount of time that has elapsed between consecutive calls. When this field reaches 0xFFFFFFFF, it wraps back to zero.

**SFTSupportLevel**

Specifies the SFT level offered by the NetWare server: 1 hot disk error fix 2 disk mirroring and transaction tracking 3 physical NetWare server mirroring

**logicalDriveCount**

Specifies the number of logical drives attached to the server. If the NetWare server supports SFT Level II or above and disks are mirrored, `logicalDriveCount` will be lower than the actual number of physical disk subsystems attached to the NetWare server. The NetWare server's operating system considers mirrored disks to be one logical drive.

**physicalDriveCount**

Specifies the number of physical disk units attached to the server.

**diskChannelTable**

Specifies the 5-byte table that indicates which disk channels exists on the server and what their drive types are. (Each channel is 1 byte.) A nonzero value in the Disk Channel Table indicates that the corresponding disk channel exists in the NetWare server. The drive types are:

```
1 = XT
2 = AT
3 = SCSI
4 = disk coprocessor
50 to 255 = Value Added Disk Drive (VADD)
```

**pendingIOCommands**

Specifies the number of outstanding disk controller commands.

**driveMappingTable**

Specifies the 32-byte table containing the primary physical drive to which each logical drive is mapped (0xFF = no such logical drive).

**driveMirrorTable**

Specifies the 32-byte table containing the secondary physical drive to which each logical drive is mapped (0xFF = no such logical drive).

**deadMirrorTable**

Specifies the 32-byte table containing the secondary physical drive to which each logical drive was last mapped (0xFF = logical drive was never mirrored). This table is used in conjunction with the Drive Mirror Table. If the entry in the Drive Mirror Table shows that a drive is not currently mirrored, the table can be used to determine which drive previously mirrored the logical drive. The Dead Mirror Table is used to remirror a logical drive after a mirror failure.

**reMirrorDriveNumber**

Specifies the physical drive number of the disk currently being remirrored (0xFF = no disk being remirrored).

**reserved**

Is currently not used.

**reMirrorCurrentOffset**

Specifies the block number that is currently being remirrored.

**SFTErrorTable**

Specifies the 60-byte table containing SFT internal error counters.

# DSK_CACHE_STATS

Returns disk caching statistics.

**Service:** Server Environment

**Defined In:** nwserver.h

## Structure

```
typedef struct
{
   nuint32   systemElapsedTime ;
   nuint16   cacheBufferCount ;
   nuint16   cacheBufferSize ;
   nuint16   dirtyCacheBuffers ;
   nuint32   cacheReadRequests ;
   nuint32   cacheWriteRequests ;
   nuint32   cacheHits ;
   nuint32   cacheMisses ;
   nuint32   physicalReadReqeusts ;
   nuint32   physicalWriteRequests ;
   nuint16   physicalReadErrors ;
   nuint16   physicalWriteErrors ;
   nuint32   cacheGetRequests ;
   nuint32   cacheFullWriteRequests ;
   nuint32   cachePartialWriteRequests ;
   nuint32   backgroundDirtyWrites ;
   nuint32   backgroundAgedWrites ;
   nuint32   totalCacheWrites ;
   nuint32   cacheAllocations ;
   nuint16   thrashingCount ;
   nuint16   LRUBlockWasDirtyCount ;
   nuint16   readBeyondWriteCount ;
   nuint16   fragmentedWriteCount ;
   nuint16   cacheHitOnUnavailCount ;
   nuint16   cacheBlockScrappedCount ;
} DSK_CACHE_STATS;
```

## Delphi Structure

```
uses calwin32

  DSK_CACHE_STATS = packed Record
    systemElapsedTime : nuint32;
    cacheBufferCount : nuint16;
    cacheBufferSize : nuint16;
    dirtyCacheBuffers : nuint16;
    cacheReadRequests : nuint32;
    cacheWriteRequests : nuint32;
    cacheHits : nuint32;
    cacheMisses : nuint32;
    physicalReadRequests : nuint32;
```

```
    physicalWriteRequests : nuint32;
    physicalReadErrors : nuint16;
    physicalWriteErrors : nuint16;
    cacheGetRequests : nuint32;
    cacheFullWriteRequests : nuint32;
    cachePartialWriteRequests : nuint32;
    backgroundDirtyWrites : nuint32;
    backgroundAgedWrites : nuint32;
    totalCacheWrites : nuint32;
    cacheAllocations : nuint32;
    thrashingCount : nuint16;
    LRUBlockWasDirtyCount : nuint16;
    readBeyondWriteCount : nuint16;
    fragmentedWriteCount : nuint16;
    cacheHitOnUnavailCount : nuint16;
    cacheBlockScrappedCount : nuint16;
  End;
```

## Fields

**systemElapsedTime**

Specifies how long the NetWare server has been up. This value is returned in units of approximately 1/18 second and is used to determine the amount of time that has elapsed between consecutive calls. when `systemElapsedTime` reaches 0xFFFFFFFF, it wraps back to zero.

**cacheBufferCount**

Specifies the number of cache buffers in the server.

**cacheBufferSize**

Specifies the number of bytes in a cache buffer.

**dirtyCacheBuffers**

Specifies the number of cache buffers in use.

**cacheReadRequests**

Specifies the number of times the cache software received a request to read data from the disk.

**cacheWriteRequests**

Specifies the number of times the cache software received a request to write data to the disk.

**cacheHits**

Specifies the number of times cache requests were serviced form existing cache blocks.

**cacheMisses**

Specifies the number of times cache requests could not be serviced form existing cache blocks.

**physicalReadReqeusts**

Specifies the number of times the cache software issued a physical read request to a disk driver. (A physical read requests reads in as much data as the cache block holds.)

**physicalWriteRequests**

Specifies the number of times the cache software issued a physical write request to a disk driver.

**physicalReadRequests**

Specifies the number of times the cache software received an error from the disk driver on a disk read request.

**physicalWriteErrors**

Specifies the number of times the cache software received an error from the disk driver on a disk write request.

**cacheGetRequests**

Specifies the number of times the cache software received a request to read information from the disk.

**cacheFullWriteRequests**

Specifies the number of times the cache software was requested to write information to disk that exactly filled one or more sectors.

**cachePartialWriteRequests**

Specifies the number of times the cache software was requested to write information to disk that did not exactly fill a sector. (Partial write requests require a disk preread.)

**backgroundDirtyWrites**

Specifies the number of times a cache block that was written to disk was completely filled with information. (The whole cache block was written.)

**backgroundAgedWrites**

Specifies the number of times the background disk write process wrote a partially filled cache block to disk. (The cache block was written to disk because the block had not been accessed for a significant period of time.)

**totalCacheWrites**

Specifies the total number of cache buffers written to disk.

**cacheAllocations**

Specifies the number of times a cache block was allocated for use.

**thrashingCount**

Specifies the number of times a cache block was not available when a cache block allocation was requested.

**LRUBlockWasDirtyCount**

Specifies the number of times the Least_Recently_Used cache block allocation algorithm reclaimed a dirty cache block.

**readBeyondWriteCount**

Specifies the number of times a file read request was received for data not yet written to disk (due to file write requests that had not yet filled the cache block). (This requires a disk preread.)

**fragmentedWriteCount**

Specifies the number of times a dirty cache block contained noncontiguous sectors of information to be written, and the skipped sectors were not preread from the disk. (Multiple disk writes were issued to write out the cache buffer.)

**cacheHitOnUnavailCount**

Specifies the number of times a cache request could be serviced from an available cache block but the cache buffer could not be used because it was in the process of being written to or read from disk.

**cacheBlockScrappedCount**

Specifies the number of times a cache block was scrapped.

# FILE_SERVER_COUNTERS

Returns information regarding the number of file packets received by the server.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   nuint16   tooManyHops ;
   nuint16   unknownNetwork ;
   nuint16   noSpaceForService ;
   nuint16   noReceiveBuffers ;
   nuint16   notMyNetwork ;
   nuint32   netBIOSProgatedCount ;
   nuint32   totalPacketsServiced ;
   nuint32   totalPacketsRouted ;
} FILE_SERVER_COUNTERS;
```

## Delphi Structure

```
uses calwin32

  FILE_SERVER_COUNTERS = packed Record
    tooManyHops : nuint16;
    unknownNetwork : nuint16;
    noSpaceForService : nuint16;
    noReceiveBuffers : nuint16;
    notMyNetwork : nuint16;
    netBIOSProgatedCount : nuint32;
    totalPacketsServiced : nuint32;
    totalPacketsRouted : nuint32;
  End;
```

## Fields

**tooManyHops**

Specifies the number of packets discarded because they had passed through more than 16 bridges without reaching their destination.

**unknownNetwork**

Specifies the number of packets discarded because their destination network was unknown to the server.

**noSpaceForService**

Is reserved (pass 0).

**noReceiveBuffers**

Specifies the number of times a packet was discarded because no buffers existed to receive it.

**notMyNetwork**

Specifies the number of received packets not destined for the server.

**netBIOSProgatedCount**

Specifies the number of NetBIOS packets received that were sent forward.

**totalPacketsServiced**

Specifies the total packets received by the server.

**totalPacketsRouted**

Specifies the number of all packets forwarded by the server.

# FSE_FILE_SYSTEM_INFO

Return file system information related to disk operation.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   FATMovedCount ;
   nuint32   FATWriteErrorCount ;
   nuint32   someoneElseDidItCount0 ;
   nuint32   someoneElseDidItCount1 ;
   nuint32   someoneElseDidItCount2 ;
   nuint32   iRanOutSomeoneElseDidItCount0 ;
   nuint32   iRanOutSomeoneElseDidItCount1 ;
   nuint32   iRanOutSomeoneElseDidItCount2 ;
   nuint32   turboFATBuildScrewedUpCount ;
   nuint32   extraUseCountNodeCount ;
   nuint32   extraExtraUseCountNodeCount ;
   nuint32   errorReadingLastFATCount ;
   nuint32   someoneElseUsingThisFileCount ;
} FSE_FILE_SYSTEM_INFO;
```

## Delphi Structure

```
uses calwin32

FSE_FILE_SYSTEM_INFO = packed Record
    FATMovedCount : nuint32;
    FATWriteErrorCount : nuint32;
    someoneElseDidItCount0 : nuint32;
    someoneElseDidItCount1 : nuint32;
    someoneElseDidItCount2 : nuint32;
    iRanOutSomeoneElseDidItCount0 : nuint32;
    iRanOutSomeoneElseDidItCount1 : nuint32;
    iRanOutSomeoneElseDidItCount2 : nuint32;
    turboFATBuildScrewedUpCount : nuint32;
    extraUseCountNodeCount : nuint32;
    extraExtraUseCountNodeCount : nuint32;
    errorReadingLastFATCount : nuint32;
    someoneElseUsingThisFileCount : nuint32;
  End;
```

## Fields

**`FATMovedCount`**

Specifies the number of times the NetWare server OS has moved the location of the FAT.

**FATWriteErrorCount**

Specifies the number of disk write errors in both the original and mirrored copy of a disk's FAT sector.

**someoneElseDidItCount0**

Specifies this is used internally by the OS.

**someoneElseDidItCount1**

Specifies this is used internally by the OS.

**someoneElseDidItCount2**

Specifies this is used internally by the OS.

**iRanOutSomeoneElseDidItCount0**

Specifies this is used internally by the OS.

**iRanOutSomeoneElseDidItCount1**

Specifies this is used internally by the OS.

**iRanOutSomeoneElseDidItCount2**

Specifies this is used internally by the OS.

**turboFATBuildScrewedUpCount**

Specifies the number of times the OS tried to allocate a Turbo FAT index but failed.

**extraUseCountNodeCount**

Specifies the number of times the OS tried to allocate a use count node for a TTS transaction but failed.

**extraExtraUseCountNodeCount**

Specifies the number of times the OS tired to allocate an additional use count node for a TTS transaction but failed.

**errorReadingLastFATCount**

Specifies the number of times the OS received an error reading the data in the last FAT.

**someoneElseUsingThisFileCount**

Specifies the number of times the OS was reading a file that another process was also reading.

# FSE_MM_OBJ_INFO

Returns media management information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   MEDIA_INFO_DEF   MediaInfo ;
   nuint32          mediaType ;
   nuint32          cartridgeType ;
   nuint32          unitSize ;
   nuint32          blockSisze ;
   nuint32          capacity ;
   nuint32          preferredUnitSize ;
   nuint8           name [64];
   nuint32          type ;
   nuint32          status ;
   nuint32          functionMask ;
   nuint32          controlMask ;
   nuint32          parentCount ;
   nuint32          siblingCount ;
   nuint32          childCount ;
   nuint32          specificInfoSize ;
   nuint32          objectUniqueID ;
   nuint32          mediaSlot ;
} FSE_MM_OBJ_INFO;
```

## Delphi Structure

```
uses calwin32

FSE_MM_OBJ_INFO = packed Record
    MediaInfo : MEDIA_INFO_DEF;
    mediaType : nuint32;
    cartridgeType : nuint32;
    unitSize : nuint32;
    blockSize : nuint32;
    capacity : nuint32;
    preferredUnitSize : nuint32;
    name : Array[0..63] Of nuint8;
    mediaManagerType : nuint32;
    status : nuint32;
    functionMask : nuint32;
    controlMask : nuint32;
    parentCount : nuint32;
    siblingCount : nuint32;
    childCount : nuint32;
    specificInfoSize : nuint32;
```

```
    objectUniqueID : nuint32;
    mediaSlot : nuint32;
  End;
```

## Fields

**MediaInfo**

Points to MEDIA_INFO_DEF (page 219).

**mediaType**

Specifies the media type of the object, as follows:

```
0  Hard disk
1  CD-ROM
2  WORM device
3  Tape device
4  Magneto-optical device
```

**cartridgeType**

Specifies the type of cartridge or magazine the device can use, as follows:

```
0x00000000  Fixed media
0x00000001  5.25 floppy
0x00000002  3.5 floppy
0x00000003  5.25 optical
0x00000004  3.5 optical
0x00000005  0.5 tape
0x00000006  0.25 tape
0x00000007  8 mm tape
0x00000008  4 mm tape
0x00000009  Bernoulli disk
```

**unitSize**

Specifies the number of bytes per sector.

**blockSize**

Specifies the maximum number of sectors that the driver can handle per I/O request.

**capacity**

Specifies the maximum number of sectors on the device.

**preferredUnitSize**

Specifies the preferred transfer unit size for the device (from 512B to 1KB for formatted devices).

**name**

Specifies the length-preceded string representing the name of the object.

**type**

Specifies the media manager database type:

```
0   AdApter Object
1   Changer object
```

```
2   Device object
4   Media object
5   Partition object
6   Slot object
7   Hotfix object
8   Mirror object
9   Parity object
10  Volume segment object
11  Volume object
12  Clone object
14  Magazine object
15  Virtual device object
FFFF  Unknown object type
```

**status**

Specifies the status mask for the object:

```
FSE_OBJECT_ACTIVATED            0x00000001
FSE_OBJECT_CREATED              0x00000002
FSE_OBJECT_SCRAMBLED            0x00000004
FSE_OBJECT_RESERVED             0x00000010
FSE_OBJECT_BEING_IDENTIFIED     0x00000020
FSE_OBJECT_MAGAZINE_LOADED      0x00000040
FSE_OBJECT_FAILURE              0x00000080
FSE_OBJECT_REMOVABLE            0x00000100
FSE_OBJECT_READ_ONLY            0x00000200
FSE_OBJECT_IN_DEVICE            0x00010000
FSE_OBJECT_ACCEPTS_MAGAZINES    0x00020000
FSE_OBJECT_IS_IN_A_CHANGER      0x00040000
FSE_OBJECT_LOADABLE             0x00080000
FSE_OBJECT_BEING_LOADED         0x00080000
FSE_OBJECT_DEVICE_LOCK          0x01000000
FSE_OBJECT_CHANGER_LOCK         0x02000000
FSE_OBJECT_REMIRRORING          0x04000000
FSE_OBJECT_SELECTED             0x08000000
```

**functionMask**

Specifies the function mask:

```
0X0001  Random read
0x0002  Random write
0x0004  Random write once
0x0008  Sequential read
0x0010  Sequential write
0x0020  Reset end of tape
0x0040  Single file mark
0x0080  Multiple file mark
0x0100  Single set mark
0x0200  Multiple set mark
0x0400  Space data blocks
0x0800  Locate data blocks
0x1000  Position partition
0x2000  Position media
```

**controlMask**

Specifies the control mask:

```
0x0001  FSE_ACTIVATE_DEACTIVE
0x0002  FSE_MOUNT_DISMOUNT
0x0004  FSE_SELECT_UNSELECT
0x0008  FSE_LOCK_UNLOCK
0x0010  FSE_EJECT
0x0020  FSE_MOVE
```

**parentCount**

Specifies the number of parent objects for the device, usually 1.

**siblingCount**

Specifies the number of sibling objects for the device.

**childCount**

Specifies the number of child objects for the device.

**specificInfoSize**

Specifies the size of the data structures that will be returned.

**objectUniqueID**

Specifies the number which identifies the device in the media manager database.

**mediaSlot**

Specifies the number of the slot the device occupies.

# FSE_SERVER_INFO

Returns information about the NetWare server.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   nuint32   replyCanceledCount ;
   nuint32   writeHeldOffCount ;
   nuint32   writeHeldOffWithDupRequest ;
   nuint32   invalidRequestTypeCount ;
   nuint32   beingAbortedCount ;
   nuint32   alreadyDoingReallocCount ;
   nuint32   deAllocInvalidSlotCount ;
   nuint32   deAllocBeingProcessedCount ;
   nuint32   deAllocForgedPacketCount ;
   nuint32   deAllocStillTransmittingCount ;
   nuint32   startStationErrorCount ;
   nuint32   invalidSlotCount ;
   nuint32   beingProcessedCount ;
   nuint32   forgedPacketCount ;
   nuint32   stillTransmittingCount ;
   nuint32   reExecuteRequestCount ;
   nuint32   invalidSequenceNumCount ;
   nuint32   duplicateIsBeingSentAlreadyCnt ;
   nuint32   sentPositiveAcknowledgeCount ;
   nuint32   sentDuplicateReplyCount ;
   nuint32   noMemForStationCtrlCount ;
   nuint32   noAvailableConnsCount ;
   nuint32   reallocSlotCount ;
   nuint32   reallocSlotCameTooSoonCount ;
} FSE_SERVER_INFO;
```

## Delphi Structure

```
uses calwin32

FSE_SERVER_INFO = packed Record
    replyCanceledCount : nuint32;
    writeHeldOffCount : nuint32;
    writeHeldOffWithDupRequest : nuint32;
    invalidRequestTypeCount : nuint32;
    beingAbortedCount : nuint32;
    alreadyDoingReallocCount : nuint32;
    deAllocInvalidSlotCount : nuint32;
    deAllocBeingProcessedCount : nuint32;
    deAllocForgedPacketCount : nuint32;
    deAllocStillTransmittingCount : nuint32;
    startStationErrorCount : nuint32;
```

```
    invalidSlotCount : nuint32;
    beingProcessedCount : nuint32;
    forgedPacketCount : nuint32;
    stillTransmittingCount : nuint32;
    reExecuteRequestCount : nuint32;
    invalidSequenceNumCount : nuint32;
    duplicateIsBeingSentAlreadyCnt : nuint32;
    sentPositiveAcknowledgeCount : nuint32;
    sentDuplicateReplyCount : nuint32;
    noMemForStationCtrlCount : nuint32;
    noAvailableConnsCount : nuint32;
    reallocSlotCount : nuint32;
    reallocSlotCameTooSoonCount : nuint32;
  End;
```

## Fields

**replyCanceledCount**

Specifies the number of replies that were cancelled because the connection was reallocated while the request was being processed.

**writeHeldOffCount**

Specifies the number of times that writes were delayed because of a pending TTS(tm) transaction or cache busy condition.

**writeHeldOffWithDupRequest**

Specifies the number of times that writes were cancelled since a duplicate request was received. (DO EITHER OF THESE REQUESTS GET WRITTEN? HOW ARE THEY PROCESSED-ORIGINAL OR DUPLICATE? HOW CAN THE GET THEM TO BE PROCESSED?)

**invalidRequestTypeCount**

Specifies the number of packets received which had an invalid request type or were received after the server was downed.

**beingAbortedCount**

Specifies the number of packets received for a connection being terminated.

**alreadyDoingReallocCount**

Specifies the number of times that a connection is requested when a connection already exists.

**deAllocInvalidSlotCount**

Specifies the number of times an attempt was made to deallocate a connection slot which was not valid.

**deAllocBeingProcessedCount**

Specifies the number of times the server was deallocated because requests were still being processed.

**deAllocForgedPacketCount**

Specifies the number of times the server was deallocated because a forget packet was received.

**deAllocStillTransmittingCount**

Specifies the number of times the server was deallocated because information was still being transmitted.

**startStationErrorCount**

Specifies the number of times the server was unable to allocate a connection for any reason.

**invalidSlotCount**

Specifies the number of requests received for an invalid connection slot.

**beingProcessedCount**

Specifies the number of times a duplicate request was received during processing of the first request.

**forgedPacketCount**

Specifies the number of suspicious invalid packets received.

**stillTransmittingCount**

Specifies the number of times a new request is received before a reply to a previous request has been sent.

**reExecuteRequestCount**

Specifies the number of times the requester did not receive the reply and the request had to be reprocessed.

**invalidSequenceNumCount**

Specifies the number of request packets the server received from a connection where the sequence number in the packet did not match the current sequence number or the next sequence number.

**duplicateIsBeingSentAlreadyCnt**

Specifies the number of times a duplicate reply was requested when the reply had already been sent.

**sentPositiveAcknowledgeCount**

Specifies the number of acknowledgments sent by the server (sent when a connection repeats a request being serviced).

**sentDuplicateReplyCount**

Specifies the number of request packets for which the server had to send a duplicate reply (only sent for requests the server cannot process).

**noMemForStationCtrlCount**

Specifies the number of times the server could not allocate memory to expand the connection table for a new connection.

**noAvailableConnsCount**

Specifies the number of times no slots were available in the connection table for a new connection.

**reallocSlotCount**

Specifies the number of times the server reallocated the same slot in the connection table for a client that logged out and relogged in.

**`reallocSlotCameTooSoonCount`**

Specifies the number of times that a request came from a client to relog in before that client had been completely logged out.

## Remarks

It is rarely possible to create suspicious packets because of faulty equipment.

If the number specified by the `forgedPacketCount` and `invalidSequenceNumCount` fields are large, it may indicate an attempt to breach network security.

Packets with bad sequence numbers are discarded.

# IPX_INFO

Returns information about the IPX protocol.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   IPXSendPacketCount ;
   nuint16   IPXMalformPacketCount ;
   nuint32   IPXGetECBRequestCount ;
   nuint32   IPXGetECBFailCount ;
   nuint32   IPXAESEventCount ;
   nuint16   IPXPostponedAESCount ;
   nuint16   IPXMaxConfiguredSocketCount ;
   nuint16   IPXMaxOpenSocketCount ;
   nuint16   IPXOpenSocketFailCount ;
   nuint32   IPXListenECBCount ;
   nuint16   IPXECBCancelFailCount ;
   nuint16   IPXGetLocalTargetFailCount ;
} IPX_INFO;
```

## Delphi Structure

```
uses calwin32

IPX_INFO = packed Record
    IPXSendPacketCount : nuint32;
    IPXMalformPacketCount : nuint16;
    IPXGetECBRequestCount : nuint32;
    IPXGetECBFailCount : nuint32;
    IPXAESEventCount : nuint32;
    IPXPostponedAESCount : nuint16;
    IPXMaxConfiguredSocketCount : nuint16;
    IPXMaxOpenSocketCount : nuint16;
    IPXOpenSocketFailCount : nuint16;
    IPXListenECBCount : nuint32;
    IPXECBCancelFailCount : nuint16;
    IPXGetLocalTargetFailCount : nuint16;
  End;
```

## Fields

**IPXSendPacketCount**

Specifies the number of IPX packets sent by the server.

**IPXMalformPacketCount**

Specifies the number of IPX packets discarded because they were malformed.

**IPXGetECBRequestCount**

Specifies the number of ECB requests.

**IPXGetECBFailCount**

Specifies the number of times an ECB was requested, but could not be supplied.

**IPXAESEventCount**

Specifies the number of AES events scheduled.

**IPXPostponedAESCount**

Specifies the number of AES events that could not be scheduled, but were placed in a waiting list.

**IPXMaxConfiguredSocketCount**

Specifies the maximum number of sockets that can be open at one time.

**IPXMaxOpenSocketCount**

Specifies the maximum number of sockets open at one time since the server was booted.

**IPXOpenSocketFailCount**

Specifies the number of times a request to open a socket failed.

**IPXListenECBCount**

Specifies the number of ECBs listening for a packet.

**IPXECBCancelFailCount**

Specifies the number of ECB listens that were cancelled.

**IPXGetLocalTargetFailCount**

Specifies the number of times the server failed to find the target.

# KNOWN_NET_INFO

Returns information about known networks.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   netIDNumber ;
   nuint16   hopsToNet ;
   nuint16   netStatus ;
   nuint16   timeToNet ;
} KNOWN_NET_INFO;
```

## Delphi Structure

```
uses calwin32

KNOWN_NET_INFO = packed Record
    netIDNumber : nuint32;
    hopsToNet : nuint16;
    netStatus : nuint16;
    timeToNet : nuint16;
  End;
```

## Fields

**netIDNumber**

Specifies the network ID number that is used by the server.

**hopsToNet**

Specifies the number of routers to cross to get to the network.

**netStatus**

Specifies the status of the network.

**timeToNet**

Specifies the number of clock ticks to the network (roundtrip).

# LAN_COMMON_INFO

Returns information about traffic on the LAN.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   notSupportedMask ;
   nuint32   totalTxPacketCount ;
   nuint32   totalRxPacketCount ;
   nuint32   noECBAvailableCount ;
   nuint32   packetTxTooBigCount ;
   nuint32   packetTxTooSmallCount ;
   nuint32   packetRxOverflowCount ;
   nuint32   packetRxTooBigCount ;
   nuint32   packetRxTooSmallCount ;
   nuint32   packetTxMiscErrorCount ;
   nuint32   packetRxMiscErrorCount ;
   nuint32   retryTxCount ;
   nuint32   checksumErrorCount ;
   nuint32   hardwareRxMismatchCount ;
   nuint32   reserved [50];
} LAN_COMMON_INFO;
```

## Delphi Structure

```
uses calwin32

  LAN_COMMON_INFO = packed Record
    notSupportedMask : nuint32;
    totalTxPacketCount : nuint32;
    totalRxPacketCount : nuint32;
    noECBAvailableCount : nuint32;
    packetTxTooBigCount : nuint32;
    packetTxTooSmallCount : nuint32;
    packetRxOverflowCount : nuint32;
    packetRxTooBigCount : nuint32;
    packetRxTooSmallCount : nuint32;
    packetTxMiscErrorCount : nuint32;
    packetRxMiscErrorCount : nuint32;
    retryTxCount : nuint32;
    checksumErrorCount : nuint32;
    hardwareRxMismatchCount : nuint32;
    reserved : Array[0..49] Of nuint32;
  End;
```

## Fields

**notSupportedMask**

Specifies a bit mask representing the fields in the statistics table. If the bit is 0, the counter is supported; if it is 1, the counter is not supported.

**totalTxPacketCount**

Specifies the total number of packets transmitted by the LAN board.

**totalRxPacketCount**

Specifies the total number of packets that were received by the LAN board.

**noECBAvailableCount**

Specifies the number of times the LAN board failed to get a receive ECB.

**packetTxTooBigCount**

Specifies the number of times the send packet was too big for this LAN board to send.

**packetTxTooSmallCount**

Specifies the number of times the send packet was too small for this LAN board to send.

**packetRxOverflowCount**

Specifies the number of times the LAN board's receive buffers overflowed.

**packetRxTooBigCount**

Specifies the number of times this LAN board could not receive a packet because the packet was too big.

**packetRxTooSmallCount**

Specifies the number of times this LAN board could not receive a packet because the packet was too small.

**packetTxMiscErrorCount**

Specifies the number of times any kind of transmit error occurred for this LAN board.

**packetRxMiscErrorCount**

Specifies the number of times any kind of receive error occurred for this LAN board.

**retryTxCount**

Specifies the number of times the LAN board retried a transmit because of failure.

**checksumErrorCount**

Specifies the number of times a checksum error occurred for this LAN board.

**hardwareRxMismatchCount**

Specifies a counter that may be incremented when a packet is received which does not pass length consistency checks.

**reserved**

Reserved for future use.

# LAN_CONFIG_INFO

Get network card (LAN card) configuration information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint8    DriverCFG_MajorVersion ;
   nuint8    DriverCFG_MinorVersion ;
   nuint8    DriverNodeAddress [6];
   nuint16   DriverModeFlags ;
   nuint16   DriverBoardNum ;
   nuint16   DriverBoardInstance ;
   nuint32   DriverMaxSize ;
   nuint32   DriverMaxRecvSize ;
   nuint32   DriverRecvSize ;
   nuint32   reserved1 [3];
   nuint16   DriverCardID ;
   nuint16   DriverMediaID ;
   nuint16   DriverTransportTime ;
   nuint8    DriverReserved [16];
   nuint8    DriverMajorVersion ;
   nuint8    DriverMinorVersion ;
   nuint16   DriverFlags ;
   nuint16   DriverSendRetries ;
   nuint32   DriverLink ;
   nuint16   DriverSharingFlags ;
   nuint16   DriverSlot ;
   nuint16   DriverIOPortsAndLengths [4];
   nuint32   DriverMemDecode0 ;
   nuint16   DriverLength0 ;
   nuint32   DriverMemDecode1 ;
   nuint16   DriverLength1 ;
   nuint8    DriverInterrupt [2];
   nuint8    DriverDMAUsage [2];
   nuint32   Reserved2 [3];
   nuint8    DriverLogicalName [18];
   nuint32   DriverLinearMem [2];
   nuint16   DriverChannelNum ;
   nuint8    DriverIOReserved [6];
} LAN_CONFIG_INFO;
```

## Delphi Structure

```
uses calwin32

  LAN_CONFIG_INFO = packed Record
    DriverCFG_MajorVersion : nuint8;
```

```
    DriverCFG_MinorVersion : nuint8;
    DriverNodeAddress : Array[0..5] Of nuint8;
    DriverModeFlags : nuint16;
    DriverBoardNum : nuint16;
    DriverBoardInstance : nuint16;
    DriverMaxSize : nuint32;
    DriverMaxRecvSize : nuint32;
    DriverRecvSize : nuint32;
    Reserved1 : Array[0..2] Of nuint32;
    DriverCardID : nuint16;
    DriverMediaID : nuint16;
    DriverTransportTime : nuint16;
    DriverReserved : Array[0..15] Of nuint8;
    DriverMajorVersion : nuint8;
    DriverMinorVersion : nuint8;
    DriverFlags : nuint16;
    DriverSendRetries : nuint16;
    DriverLink : nuint32;
    DriverSharingFlags : nuint16;
    DriverSlot : nuint16;
    DriverIOPortsAndLengths : Array[0..3] Of nuint16;
    DriverMemDecode0 : nuint32;
    DriverLength0 : nuint16;
    DriverMemDecode1 : nuint32;
    DriverLength1 : nuint16;
    DriverInterrupt : Array[0..1] Of nuint8;
    DriverDMAUsage : Array[0..1] Of nuint8;
    Reserved2 : Array[0..2] Of nuint32;
    DriverLogicalName : Array[0..17] Of nuint8;
    DriverLinearMem : Array[0..1] Of nuint32;
    DriverChannelNum : nuint16;
    DriverIOReserved : Array[0..5] Of nuint8;
  End;
```

## Fields

**DriverCFG_MajorVersion**

Specifies the Novell® defined major version number of the configuration table.

**DriverCFG_MinorVersion**

Specifies the Novell defined minor version of the configuration table.

**DriverNodeAddress**

Specifies the node address of the LAN board.

**DriverModeFlags**

Specifies the mode supported by the driver:

0x0001 Specifies whether the driver was real or a dummy; set to 1.

0x0002 Specifies if the driver uses DMA.

0x0004 Specifies to routers to pass router table changes when they occur, rather than forwarding all RIP and SAP packets; set only if but 4 is set.

0x0008 Specifies if the driver supports multicasting.

0x0010 Specifies if the driver can bind with a protocol stack without providing a network number.

0x0030 Specifies if the driver supports raw sends, no prepending any hardware header.

0x0400 Specifies if the HSM can handle fragmented RCBs.

0x2000 Specifies if the HSM can handle promiscuous RCBs.

0xC000 Specifies the driver node address, as follows:

```
00 Format is unspecified; the node address is assumed to be in the
native format of the physical layer.
01 Illegal combination
10 Driver node address is canonical
11 Driver node address is noncanonical
```

**DriverBoardNum**

Specifies the logical board number (1-255) assigned to the LAN board by the LSL(tm) service.

**DriverBoardInstance**

Specifies the number of the physical card the logical board is using.

**DriverMaxSize**

Specifies the maximum send or receive packet size in bytes the board can handle.

**DriverMaxRecvSize**

Specifies the maximum packet size in bytes that the LAN board can receive.

**DriverRecvSize**

Specifies the maximum packet size in bytes a protocol stack can send or receive using this board.

**reserved1**

Is reserved (pass zero).

**DriverCardID**

Specifies the number assigned to the LAN board by IMSP.

**DriverMediaID**

Specifies the number identifying the link-level envelope used by the MLID.

**DriverTransportTime**

Specifies the time in ticks to transmit a 576-byte packet.

**DriverReserved**

Reserved for future use (currently set to zero).

**DriverMajorVersion**

Specifies the major version number of the MLID.

**DriverMinorVersion**

Specifies the minor version number of the MLID.

**DriverFlags**

Specifies a bit map showing the architecture supported by the MLID:

| | |
|---|---|
| 0x0001 | EISA |
| 0x0002 | ISA |
| 0x0004 | MCA |
| 0x0100 | Hub management |
| 0x0600 | Multicast filtering and format: 00 LAN medium defaults 01 Illegal combination |

The following bits are set if the board can share:

| | |
|---|---|
| 0x0020 | Primary interrupt |
| 0x0040 | Secondary interrupt |
| 0x0080 | DMA channel 0 |
| 0x0100 | DMA channel 1 |

The following bits are set if:

| | |
|---|---|
| 0x0200 | A command line information string to place in AUTOEXEC.NCF is available. |
| 0x0400 | To prevent default information from the AUTOEXEC.NCF, this bit overrides the setting of bit 9. |

**DriverSendRetries**

Contains the number of times that the MLID retries send events before aborting the send.

**DriverLink**

Used by the LSL.

**DriverSharingFlags**

Contains a bit map defining the sharing abilities of the MLID.

**DriverSlot**

Specifies the slot number of the board if installed in MCA or EISA machine; otherwise it is 0.

**DriverIOPortsAndLengths**

Each WORD is defined below:

| | |
|---|---|
| Word 1 | Primary base I/O port |
| Word 2 | Number of I/O ports beginning with primary base I/O port |
| Word 3 | Secondary base I/O port |
| Word 4 | Number of I/O ports beginning with secondary base I/O port |

**DriverMemDecode0**

Specifies the absolute primary memory address that the LAN board uses.

**DriverLength0**

Specifies the amount of memory in paragraphs the board uses starting at `DriverMemDecode0`

**DriverMemDecode1**

Specifies the absolute secondary memory address the board uses.

**DriverLength1**

Specifies the amount of memory in paragraphs the board uses, starting at `DriverMemDecode1`.

**DriverInterrupt**

Specifies the primary interrupt in the first byte; secondary interrupt in the secondary byte. FFh means not used.

**DriverDMAUsage**

Specifies the primary DMA channel used in the board in the first byte; secondary DMA channel in the second byte. FFh means not used.

**Reserved2**

Specifies the logical name of the LAN driver, given at load time.

**DriverLogicalName**

Specifies the logical name of the LAN driver, given at load time.

**DriverLinearMem**

Specifies the addresses of `DriverMemDecode0` and `DriverMemDecode1` in the first and second LONGS.

**DriverChannelNum**

Specifies the multichannel adapters. It holds the channel number of the NIC to use.

**DriverIOReserved**

Reserved for the LSL.

# LSL_INFO

Get LSL (Link Support Layer) information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   rxBufs ;
   nuint32   rxBufs75PerCent ;
   nuint32   rxBufsCheckedOut ;
   nuint32   rxBufMaxSize ;
   nuint32   maxPhysicalSize ;
   nuint32   lastTimeRxBufAllocated ;
   nuint32   maxNumsOfProtocols ;
   nuint32   maxNumsOfMediaTypes ;
   nuint32   totalTXPackets ;
   nuint32   getECBBfrs ;
   nuint32   getECBFails ;
   nuint32   AESEventCounts ;
   nuint32   postponedEvents ;
   nuint32   ECBCxlFails ;
   nuint32   validBfrsReused ;
   nuint32   enqueuedSendCount ;
   nuint32   totalRXPackets ;
   nuint32   unclaimedPackets ;
   nuint8    StatisticsTableMajorVersion ;
   nuint8    StatisticsTableMinorVersion ;
} LSL_INFO;
```

## Delphi Structure

```
uses calwin32

 LSL_INFO = packed Record
    rxBufs : nuint32;
    rxBufs75PerCent : nuint32;
    rxBufsCheckedOut : nuint32;
    rxBufMaxSize : nuint32;
    maxPhysicalSize : nuint32;
    lastTimeRxBufAllocated : nuint32;
    maxNumsOfProtocols : nuint32;
    maxNumsOfMediaTypes : nuint32;
    totalTXPackets : nuint32;
    getECBBfrs : nuint32;
    getECBFails : nuint32;
    AESEventCounts : nuint32;
    postponedEvents : nuint32;
    ECBCxlFails : nuint32;
```

```
    validBfrsReused : nuint32;
    enqueuedSendCount : nuint32;
    totalRXPackets : nuint32;
    unclaimedPackets : nuint32;
    StatisticsTableMajorVersion : nuint8;
    StatisticsTableMinorVersion : nuint8;
  End;
```

## Fields

**rxBufs**

Specifies the total number of LSL receive buffers.

**rxBufs75PerCent**

Specifies the number of LSL receive buffers that must be in use before a warning message is issued that buffers are getting low.

**rxBufsCheckedOut**

Specifies the number of LSL buffers in use.

**rxBufMaxSize**

Specifies the size of the data portion of the ECBs in bytes.

**maxPhysicalSize**

Specifies the total size of the ECB in bytes.

**lastTimeRxBufAllocated**

Specifies the last time in ticks a buffer was checked out.

**maxNumsOfProtocols**

Specifies the number of protocol stacks supported by the OS.

**maxNumsOfMediaTypes**

Specifies the number of frame types supported by the OS.

**totalTXPackets**

Specifies the number of packet transmit requests.

**getECBBfrs**

Contains the number of ECBs that were requested.

**getECBFails**

Specifies the number of times an ECB request failed.

**AESEventCounts**

Specifies the total number of AES events that have been processed.

**postponedEvents**

Specifies the total number of AES events postponed because of critical sections.

**ECBCxlFails**

Specifies the number of AES cancel requests that failed because the event was not found on the AES list.

**validBfrsReused**

Specifies the number of ECBs in the hold queue that were reused before they were removed from the hold queue.

**enqueuedSendCount**

Specifies the number of send events in the queue that have occurred.

**totalRXPackets**

Specifies the total number of received incoming packets.

**unclaimedPackets**

Specifies the total number of unclaimed incoming packets.

**StatisticsTableMajorVersion**

Contains the major version of the LSL statistics table.

**StatisticsTableMinorVersion**

Contains the minor version of the LSL statistics table.

# MEDIA_INFO_DEF

Returns information on the media manager object.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint8    label [64];
   nuint32   identificationType ;
   nuint32   identificationTimeStamp ;
} MEDIA_INFO_DEF;
```

## Delphi Structure

```
uses calwin32

 MEDIA_INFO_DEF = packed Record
    mediaLabel : Array[0..63] Of nuint8;
    identificationType : nuint32;
    identificationTimeStamp : nuint32;
  End;
```

## Fields

**label**

Specifies the name of the object.

**identificationType**

Specifies the Novell assigned number for the object.

**identificationTimeStamp**

Specifies the DOS timestamp of the object.

# MLID_BOARD_INFO

Contains information about each Multiple Link Interface Driver (MLID).

**Service:** Server Environment

**Defined In:** nwfse.h

## Syntax

```
typedef struct
{
   nuint32   protocolBoardNum;
   nuint16   protocolNumber;
   nuint8    protocolID[6];
   nuint8    protocolName[16];
} MLID_BOARD_INFO;
```

## Delphi Syntax

```
Type
 MLID_BOARD_INFO = packed Record
    protocolBoardNum : nuint32;
    protocolNumber : nuint16;
    protocolID: Array[0..5] of nuint8;
    protocolName : Array[0..15] of nuint8;
 End;
```

## Fields

**`protocolBoardNum`**

Specifies the board number the protocol is using.

**`protocolNumber`**

Specifies the protocol number.

**`protocolID`**

Specifies the protocol ID.

**`protocolName`**

Specifies the protocol name as a length-preceded string.

## Remarks

`protocolNumber` and `protocolID` can have the following values:

| Name & Ethernet Frame Type | Number | ID (from Hi-Lo) | | | | | |
|---|---|---|---|---|---|---|---|
| | | Byte 5 | Byte 4 | Byte 3 | Byte 2 | Byte 1 | Byte 0 |
| IPX: 802.2 | 0 | 0xE0 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 |

| Name & Ethernet Frame Type | Number | ID (from Hi-Lo) | | | | | |
|---|---|---|---|---|---|---|---|
| | | Byte 5 | Byte 4 | Byte 3 | Byte 2 | Byte 1 | Byte 0 |
| IPX: 802.3 | | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 |
| IPX: Ethernet II | | 0x37 | 0x81 | 0x00 | 0x00 | 0x00 | 0x00 |
| IP | 1 | 0x00 | 0x08 | 0x00 | 0x00 | 0x00 | 0x00 |
| ARP | 2 | 0x06 | 0x08 | 0x00 | 0x00 | 0x00 | 0x00 |

# NETWARE_PRODUCT_VERSION

Returns NetWare product version information (major version, minor version and revision).

**Service:** Server Environment

**Defined In:** nwserver.h

## Structure

```
typedef struct
{
   nuint16   majorVersion;
   nuint16   minorVersion;
   nuint16   revision;
} NETWARE_PRODUCT_VERSION;
```

## Delphi Structure

```
uses calwin32

  NETWARE_PRODUCT_VERSION = Record
    majorVersion : nuint16;
    minorVersion : nuint16;
    revision : nuint16;
  End;
```

## Fields

**majorVersion**

Specifies the major version number of the NetWare product.

**minorVersion**

Specifies the minor version number of the NetWare product.

**revision**

Specifies the version revision letter of the NetWare product. The revision letter can be expressed as a number where a = 0, b = 1, and so forth.

# NLM_INFO

Returns information about an NLM.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   identificationNum ;
   nuint32   flags ;
   nuint32   type ;
   nuint32   parentID ;
   nuint32   majorVersion ;
   nuint32   minorVersion ;
   nuint32   revision ;
   nuint32   year ;
   nuint32   month ;
   nuint32   day ;
   nuint32   allocAvailableBytes ;
   nuint32   allocFreeCount ;
   nuint32   lastGarbageCollection ;
   nuint32   messageLanguage ;
   nuint32   numOfReferencedPublics ;
} NLM_INFO;
```

## Delphi Structure

```
uses calwin32

 NLM_INFO = packed Record
    identificationNum : nuint32;
    flags : nuint32;
    NLMtype : nuint32;
    parentID : nuint32;
    majorVersion : nuint32;
    minorVersion : nuint32;
    revision : nuint32;
    year : nuint32;
    month : nuint32;
    day : nuint32;
    allocAvailableBytes : nuint32;
    allocFreeCount : nuint32;
    lastGarbageCollection : nuint32;
    messageLanguage : nuint32;
    numOfReferencedPublics : nuint32;
  End;
```

## Fields

**identificationNum**

Specifies the number assigned to the NLM when it was loaded.

**flags**

Specifies a bit mask. Bits are defined as follows:

```
0x0001 = REENTRANT
0x0002 = MULTIPLE
0x0004 = SYNCHRONIZE
0x0008 = PSEUDOPREEMPTION
```

**type**

Specifies the type:

```
0 = NLM_GENERIC
1 = LAN_DRIVER
2 = DSK_DRIVER
3 = NAM_SPACE
4 = NLM_UTILITY
5 = MIRRORED_SERVER_LINK
6 = NLM_OS
7 = NLM_PAGED_HIGH_OS
8 = HOST_ADAPTER_MODULE
9 = CUSTOM_DEVICE_MODULE
10 = NLM_FILE_SYSTEM
11 = NLM_REAL_MODE
12 = GHOST_TYPE
13 = SMP_NORMAL_TYPE
14 = NIOS_TYPE_NLM
15 = CIOS_CAD_TYPE
16 = CIOS_CLS_TYPE
20 through 32 = NICI (Novell International Cryptographic
Infrastructure)
```

**parentID**

Specifies the number of the NLM that caused this NLM to be loaded.

**majorVersion**

Specifies the major version of the NLM.

**minorVersion**

Specifies the minor version of the NLM.

**revision**

Specifies the revision letter of the NLM.

**year**

Specifies the timestamp of the NLM.

**month**

Specifies the timestamp of the NLM.

**day**

Specifies the timestamp of the NLM.

**allocAvailableBytes**

Specifies the bytes available for allocation by the NLM.

**allocFreeCount**

Specifies the number of bytes freed that can be reclaimed.

**lastGarbageCollection**

Specifies the last time garbage collection was done for the NLM.

**messageLanguage**

Specifies the number representing the language the NLM uses.

**numOfReferencedPublics**

Specifies the number of external symbols referenced by the NLM.

# NWFSE_ACCT_INFO

Returns Server Environment accounting information. Used by NWGetServerConnInfo (page 143).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   holdTime;
   nuint32   holdAmt;
   nuint32   chargeAmt;
   nuint32   heldConnectTimeInMinutes;
   nuint32   heldRequests;
   nuint8    heldBytesRead[6];
   nuint8    heldBytesWritten[6];
} NWFSE_ACCT_INFO;
```

## Delphi Syntax

```
uses calwin32

 NWFSE_ACCT_INFO  = packed RECORD
   holdTime : nuint32;
   holdAmt : nuint32;
   chargeAmt : nuint32;
   heldConnectTimeInMinutes : nuint32;
   heldRequests : nuint32;
   heldBytesRead : Array[1..6] of nuint8;
   heldBytesWritten : Array[1..6] of nuint8;
 End;
```

## Fields

**holdTime**

Specifies the amount of time that the specified amount will be held before being charged to the object's account balance.

**holdAmt**

Specifies the amount to be held against an object's account balance.

**chargeAmt**

Specifies the amount to be charged to an object's account balance.

**heldConnectTimeInMinutes**

Specifies the connect time (in minutes) that is held before being charged to an object's account.

**heldRequests**

Specifies the number of requests held for accounting purposes.

**`heldBytesRead`**

Specifies the number of bytes the user read that have a hold on them for accounting purposes.

**`heldBytesWritten`**

Specifies the number of bytes the user wrote that have a hold on them for accounting purposes.

## Remarks

The hold fields are designed to hold information as the server is reserving and calculating how much of the object's account balance will be charged. Once the charge is made against the object's account, the hold fields are cleared.

# NWFSE_ACTIVE_CONN_LIST

Returns the Active Connection List by type. Used by NWGetActiveConnListByType (page 45).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint8                     activeConnBitList [512];
} NWFSE_ACTIVE_CONN_LIST;
```

## Delphi Structure

```
uses calwin32

  NWFSE_ACTIVE_CONN_LIST = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding : nuint16;
    activeConnBitList : Array[0..511] Of nuint8;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Reserved for future use.

**activeConnBitList**

Indicates active connections. An array of 512 bytes is returned where a bit is set for each active connection. The connection number is determined by its position in the array.

# NWFSE_ACTIVE_LAN_BOARD_LIST

Returns a list of active LAN boards in the server. Used by NWGetActiveLANBoardList (page 47).

**Service:** Server Environment

**Defined In:** nwfserver.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo:
   nuint16                    reserved ;
   nuint32                    MaxNumOfLANs ;
   nuint32                    LANLoadedCount ;
   nuint32                    boardNums [FSE_MAX_NUM_OF_LANS];
} NWFSE_ACTIVE_LAN_BOARD_LIST;
```

## Delphi Structure

```
uses calwin32

 NWFSE_ACTIVE_LAN_BOARD_LIST = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    MaxNumOfLANs : nuint32;
    LANLoadedCount : nuint32;
    boardNums : Array[0..FSE_MAX_NUM_OF_LANS-1] Of nuint32;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Reserved for future use.

**MaxNumOfLANs**

Specifies the maximum number of LAN boards that can be used on the server.

**LANLoadedCount**

contains the number of LAN boards returned by this call to SSGetActiveLANBoardList (page 482). To retrieve the rest of the board numbers, call this function again, using the total number of items returned by all previous calls to SSGetActiveLANBoardList plus 1 as startNumber.

**boardNums**

Contains the first LAN board number. The first number is followed by board numbers for each LAN board.

# NWFSE_ACTIVE_STACKS

Returns information about active protocol stacks. Used by NWGetActiveProtocolStacks (page 49).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo:
   nuint16                    reserved ;
   nuint32                    maxNumOfStacks ;
   nuint32                    stackCount ;
   nuint32                    nextStartNum ;
   STACK_INFO                 stackInfo [FSE_MAX_NUM_OF_STACKINFO];
} NWFSE_ACTIVE_STACKS;
```

## Delphi Structure

```
uses calwin32

 NWFSE_ACTIVE_STACKS = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    maxNumOfStacks : nuint32;
    stackCount : nuint32;
    nextStartNum : nuint32;
    stackInfo : Array[0.. FSE_MAX_NUM_OF_STACKINFO -1] Of STACK_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Reserved for future use.

**maxNumOfStacks**

Specifies the total number of protocol stacks.

**stackCount**

Specifies the number of STACK_INFO (page 311) structures in the buffer.

**nextStartNum**

Is the start number to use on subsequent calls.

**stackInfo**

Contains the first ProtocolStackInfo structure in the buffer. The fields of this structure are defined as follows:

| | |
|---|---|
| `stackNumber` field | Contains the protocol number. |
| `stackName` field | The contains the protocol name. |

# NWFSE_AUTH_INFO

Returns Server Environment authentication information. Used by NWGetServerConnInfo (page 143).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   loginStatus;
   nuint32   loginPrivileges;
} NWFSE_AUTH_INFO;
```

## Delphi Syntax

```
Type
   NWFSE_AUTH_INFO = packed RECORD
      loginStatus     : nuint32;
      loginPrivileges : nuint32;
   End;
```

## Fields

**`loginStatus`**

Specifies the login status:

| Number | Constant |
|---|---|
| 0x00000001 | LOGGED_IN |
| 0x00000002 | BEING_ABORTED |
| 0x00000010 | MAC_STATION |
| 0x00000020 | AUTHENTICATED_TEMPORARY |
| 0x00000100 | LOGOUT_IN_PROGRESS |
| 0x00000200 | INTERNAL_LOGIN |
| 0x00000400 | BINDERY_CONNECTION |

**`loginPrivileges`**

Specifies the access privileges the logged in user possesses:

0x1 Supervisor privileges
0x2 Console operator privileges
0x4 Auditor privileges

# NWFSE_CACHE_INFO

Returns Server Environment cache information. Used by NWGetCacheInfo (page 51).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved ;
   CACHE_COUNTERS             cacheCounters ;
   CACHE_MEM_COUNTERS         cacheMemCounters ;
   CACHE_TREND_COUNTERS       cacheTrendCounters ;
   CACHE_INFO                 cacheInformation ;
} NWFSE_CACHE_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_CACHE_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    cacheCounters : CACHE_COUNTERS;
    cacheMemCounters : CACHE_MEM_COUNTERS;
    cacheTrendCounters : CACHE_TREND_COUNTERS;
    cacheInformation : CACHE_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Points to SERVER_AND_VCONSOLE_INFO.

**reserved**

Is reserved for future use.

**cacheCounters**

Points to CACHE_COUNTERS.

**cacheMemCounters**

Points to CACHE_MEM_COUNTERS.

**cacheTrendCounters**

Points to CACHE_TREND_COUNTERS.

**cacheInformation**

Points to CACHE_INFO.

# NWFSE_CPU_INFO

Returns Server Environment CPU information. Used by NWGetCPUInfo (page 53).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    numOfCPUs ;
   CPU_INFO                   CPUInfo ;
} NWFSE_CPU_INFO;
```

## Delphi Structure

```
uses calwin32

NWFSE_CPU_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    numOfCPUs : nuint32;
    CPUInfo : CPU_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Points to SERVER_AND_VCONSOLE_INFO.

**reserved**

Is reserved (pass 0).

**numOfCPUs**

Specifies the number of CPUs in the server.

**CPUInfo**

Points to the CPU_INFO structure.

# NWFSE_DIR_CACHE_INFO

Returns Directory cache information. Used by NWGetDirCacheInfo (page 55).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   DIR_CACHE_INFO             dirCacheInfo ;
} NWFSE_DIR_CACHE_INFO;
```

## Delphi Structure

```
uses calwin32

 NWFSE_DIR_CACHE_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    dirCacheInfo : DIR_CACHE_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Reserved for future use.

**dirCacheInfo**

Contains a DIR_CACHE_INFO (page 184) structure.

# NWFSE_FILE_SERVER_INFO

Returns file server information. Used by NWGetFileServerInfo (page 65).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    NCPStationsInUseCount ;
   nuint32                    NCPPeakStationsInUseCount ;
   nuint32                    numOfNCPRequests ;
   nuint32                    serverUtilization ;
   FSE_SERVER_INFO            ServerInfo ;
   FILE_SERVER_COUNTERS       fileServerCounters ;
} NWFSE_FILE_SERVER_INFO;
```

## Delphi Structure

```
uses calwin32

NWFSE_FILE_SERVER_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    NCPStationsInUseCount : nuint32;
    NCPPeakStationsInUseCount : nuint32;
    numOfNCPRequests : nuint32;
    serverUtilization : nuint32;
    ServerInfo : FSE_SERVER_INFO;
    fileServerCounters : FILE_SERVER_COUNTERS;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Contains console version information and the time elapsed since the server was brought up. For more information, see SERVER_AND_VCONSOLE_INFO (page 306).

**reserved**

Reserved for future use.

**NCPStationsInUseCount**

Specifies the number of workstations connected to the server.

**NCPPeakStationsInUseCount**

Specifies the maximum number of workstations connected at one time since the server was brought up.

**numOfNCPRequests**

Specifies the number of NCP requests received by the server since it was brought up.

**serverUtilization**

Specifies the current percentage of CPU utilization for the server.

**ServerInfo**

Specifies the NetWare server statistics.

**fileServerCounters**

Specifies the NetWare server statistics.

# NWFSE_FILE_SYSTEM_INFO

Returns NetWare File Systems information. Used by NWGetNetWareFileSystemsInfo (page 112).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   FSE_FILE_SYSTEM_INFO       fileSystemInfo ;
} NWFSE_FILE_SYSTEM_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_FILE_SYSTEM_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    fileSystemInfo : FSE_FILE_SYSTEM_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Reserved for future use.

**fileSystemInfo**

Pointer to FSE_FILE_SYSTEM_INFO (page 196).

# NWFSE_GARBAGE_COLLECTION_INFO

Returns information about failed requests. Used by NWGetGarbageCollectionInfo (page 79).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    failedAllocRequestCount ;
   nuint32                    numOfAllocs ;
   nuint32                    noMoreMemAvailableCount ;
   nuint32                    numOfGarbageCollections ;
   nuint32                    garbageFoundSomeMem ;
   nuint32                    garbageNumOfChecks ;
} NWFSE_GARBAGE_COLLECTION_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_GARBAGE_COLLECTION_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    failedAllocRequestCount : nuint32;
    numOfAllocs : nuint32;
    noMoreMemAvailableCount : nuint32;
    numOfGarbageCollections : nuint32;
    garbageFoundSomeMem : nuint32;
    garbageNumOfChecks : nuint32;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

**reserved**

Is reserved (pass zero).

**failedAllocRequestCount**

Specifies the number of memory allocations that failed since the server was brought up.

**numOfAllocs**

Specifies the number of memory allocations made since the server was brought up.

**noMoreMemAvailableCount**

Specifies the number of times that allocation failed because there was no memory available since the server was brought up.

**numOfGarbageCollections**

Specifies the number of times garbage collection was invoked since the server was brought up.

**garbageFoundSomeMem**

Specifies the number of times garbage collection reclaimed memory since the server was brought up.

**garbageNumOfChecks**

Specifies the number of times garbage collection checked for memory since the server was brought up.

# NWFSE_GENERAL_ROUTER_SAP_INFO

Returns router and SAP information. Used by NWGetGeneralRouterAndSAPInfo (page 81).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo
   nuint16                    reserved ;
   nuint32                    internalRIPSocket ;
   nuint32                    internalRouterDownFlag ;
   nuint32                    trackOnFlag ;
   nuint32                    externalRouterActiveFlag ;
   nuint32                    internalSAPSocketNumber ;
   nuint32                    replyToNearestServerFlag ;
} NWFSE_GENERAL_ROUTER_SAP_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_GENERAL_ROUTER_SAP_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    internalRIPSocket : nuint32;
    internalRouterDownFlag : nuint32;
    trackOnFlag : nuint32;
    externalRouterActiveFlag : nuint32;
    internalSAPSocketNumber : nuint32;
    replyToNearestServerFlag : nuint32;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Reserved for future use.

**internalRIPSocket**

Specifies the router socket number.

**internalRouterDownFlag**

Specifies whether the internal router is up or down.

**trackOnFlag**

Specifies whether router tracking is active (the console operator issued the TRACK ON console command).

**externalRouterActiveFlag**

Specifies whether an external router is active.

**internalSAPSocketNumber**

Specifies the number of the socket that receives SAP packets.

**replyToNearestServerFlag**

Specifies whether the server will respond to GetNearestServer.

# NWFSE_IPXSPX_INFO

Returns information about IPX/SPX use on a server. Used by NWGetIPXSPXInfo (page 83).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   IPX_INFO                   IPXInfo ;
   SPX_INFO                   SPXInfo ;
} NWFSE_IPXSPX_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_IPXSPX_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    IPXInfo : IPX_INFO;
    SPXInfo : SPX_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Reserved for future use.

**IPXInfo**

Contains a IPX_INFO (page 206) structure. This structure is defined in NWSERVST.H.

**SPXInfo**

Contains a SPX_INFO (page 308)structure. This structure is defined in NWSERVST.H.

# NWFSE_KNOWN_NETWORKS_INFO

Returns information about known networks. Used by NWGetKnownNetworksInfo (page 85).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved ;
   nuint32                    numberOfEntries ;
   KNOWN_NET_INFO             knownNetInfo [51];
} NWFSE_KNOWN_NETWORKS_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_KNOWN_NETWORKS_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    numberOfEntries : nuint32;
    knownNetInfo : Array[0..50] Of KNOWN_NET_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Reserved for future use.

**numberOfEntries**

Contains the number of entries for which information is returned.

**knownNetInfo**

Specifies the first KNOWN_NET_INFO (page 208) structure.

# NWFSE_KNOWN_SERVER_INFO

Returns server information. Used by NWGetKnownServersInfo (page 87).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleIInfo;
   nuint16                    reserved ;
   nuint32                    numberOfEntries ;
   nuint8                     data [512];
} NWFSE_KNOWN_SERVER_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_KNOWN_SERVER_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    numberOfEntries : nuint32;
    data : Array[0..511] Of nuint8;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved (pass 0).

**numberOfEntries**

Specifies the number of entries.

**data**

Specifies an array containing the following fields: SERVER_INFO=RECORD network Address: Array[0..3] of BYTE; nodeAddress: Array[0..5] of BYTE; socketAddress: nuint16; HopsToServer: nuint16; ServerName: Array[0..47] of char8; (#0 terminated) END;

The next field starts immediately after the trailing #0 of the last `ServerName` field.

# NWFSE_LAN_COMMON_COUNTERS_INFO

Returns information on LAN common counters. Used by NWGetLANCommonCountersInfo (page 89).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint8                     statisticsMajorVersion ;
   nuint8                     statisticsMinorVersion ;
   nuint32                    numberOfGenericCounters ;
   nuint32                    numberOfCounterBlocks ;
   nuint32                    customVariableCount ;
   nuint32                    NextCounterBlock ;
   LAN_COMMON_INFO            LANCommonInfo ;
} NWFSE_LAN_COMMON_COUNTERS_INFO;
```

## Delphi Structure

```
uses calwin32

 NWFSE_LAN_COMMON_COUNTERS_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    statisticsMajorVersion : nuint8;
    statisticsMinorVersion : nuint8;
    numberOfGenericCounters : nuint32;
    numberOfCounterBlocks : nuint32;
    customVariableCount : nuint32;
    NextCounterBlock : nuint32;
    LANCommonInfo : LAN_COMMON_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**statisticsMajorVersion**

Specifies the major version number of the statistics table.

**statisticsMinorVersion**

Specifies the minor version number of the statistics table.

**numberOfGenericCounters**

Specifies the total number of LAN common counters.

**numberOfCounterBlocks**

Specifies the number of blocks used by LAN common counters by the LAN board.

**customVariableCount**

Specifies the number of custom counters for this LAN board.

**NextCounterBlock**

Specifies the value to be passed in block numbers to the NWGetLANCommonCountersInfo function.

**LANCommonInfo**

Points to the LAN_COMMON_INFO structure containing information about the LAN board.

## Remarks

When 0 is returned in the `NextCounterBlock` field, all common counters have been returned.

# NWFSE_LAN_CONFIG_INFO

Returns LAN configuration information. Used by NWGetLANConfigInfo (page 91).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   LAN_CONFIG_INFO            LANConfigInfo ;
} NWFSE_LAN_CONFIG_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_LAN_CONFIG_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    LANConfigInfo : LAN_CONFIG_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved (pass 0).

**LANConfigInfo**

Contains LAN configuration information.

# NWFSE_LAN_CUSTOM_INFO

Returns information on LAN custom counters. Used by NWGetLANCustomCountersInfo (page 93).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    numCustomVar ;
   nuint8                     customInfo [512];
} NWFSE_LAN_CUSTOM_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_LAN_CUSTOM_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    numCustomVar : nuint32;
    customInfo : Array[0..511] Of nuint8;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved (pass 0).

**numCustomVar**

Specifies the value of the custom counter.

**customInfo**

Specifies the description of the custom counter.

# NWFSE_LOADED_MEDIA_NUM_LIST

Returns a list of loaded media numbers. Used by NWGetLoadedMediaNumList (page 95).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    maxMediaTypes ;
   nuint32                    mediaListCount ;
   nuint32                    mediaList [FSE_MEDIA_LIST_MAX];
} NWFSE_LOADED_MEDIA_NUM_LIST;
```

## Delphi Structure

```
uses calwin32

  NWFSE_LOADED_MEDIA_NUM_LIST = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    maxMediaTypes : nuint32;
    mediaListCount : nuint32;
    mediaList : Array[0.. FSE_MEDIA_LIST_MAX -1] Of nuint32;
  End;
```

## Fields

**`serverTimeAndVConsoleInfo`**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**`reserved`**

Is reserved (pass 0).

**`maxMediaTypes`**

Specifies the maximum number of media allowed.

**`mediaListCount`**

Specifies the number of valid IDs returned in the `mediaList` parameter.

**`mediaList`**

Specifies the ID numbers of the returned media.

# NWFSE_LOCK_INFO

Returns Server Environment locking information. Used by NWGetServerConnInfo (page 143).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint8    logicalLockThreshold;
   nuint8    recordLockThreshold;
   nuint16   fileLockCount;
   nuint16   recordLockCount;
} NWFSE_LOCK_INFO;
```

## Delphi Syntax

```
uses calwin32

  NWFSE_LOCK_INFO = packed RECORD
    logicalLockThreshold : nuint8;
    recordLockThreshold  : nuint8;
    fileLockCount        : nuint16;
    recordLockCount      : nuint16;
  End;
```

## Fields

**logicalLockThreshold**

Specifies the maximum number of logical locks a user can have.

**recordLockThreshold**

Specifies the maximum number of record locks the user can have.

**fileLockCount**

Specifies the number of files the user locked.

**recordLockCount**

Specifies the number of records the user locked.

# NWFSE_LOGIN_NAME

Returns the login name of the object. Used by NWGetServerConnInfo (page 143).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   loginObjectType;
   nuint8    loginNameLen;
   pnuint8   loginName;
} NWFSE_LOGIN_NAME;
```

## Delphi Syntax

```
uses calwin32

  NWFSE_LOGIN_NAME  = packed RECORD
     loginObjectType : nuint32;
     loginNameLen : nuint8;
     loginName : pnuint8;
  End;
```

## Fields

**loginObjectType**

Specifies the type of the logged in object (user, group, server, etc.).

**loginNameLen**

Specifies the length of the login name string.

**loginName**

Points to the string containing the name of the logged in object.

# NWFSE_LOGIN_TIME

Returns the login time of the object. Used by NWGetServerConnInfo (page 143).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint8    loginTime[7];
   nuint32   loginExpirationTime;
} NWFSE_LOGIN_TIME;
```

## Delphi Syntax

```
uses calwin32

  NWFSE_LOGIN_TIME  = packed RECORD
     loginTime : Array[1..7] of nuint8;
     loginExpirationTime : nuint32;
  End;
```

## Fields

**`loginTime`**

Specifies the time the user logged in.

**`loginExpirationTime`**

Specifies the expiration time of the login.

# NWFSE_LSL_INFO

Returns LSL information. Used by NWGetLSLInfo (page 97).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   LSL_INFO                   LSLInfo ;
} NWFSE_LSL_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_LSL_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding : nuint16;
    LSLInfo : LSL_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved (pass 0).

**LSLInfo**

Points to the LSL_INFO structure containing the LSL information.

# NWFSE_LSL_LOGICAL_BOARD_STATS

Returns statistics concerning LSL boards. Used by NWGetLSLLogicalBoardStats (page 99).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved0 ;
   nuint32                    LogTtlTxPackets ;
   nuint32                    LogTtlRxPackets ;
   nuint32                    LogUnclaimedPackets ;
   nuint32                    reserved1 ;
} NWFSE_LSL_LOGICAL_BOARD_STATS;
```

## Delphi Structure

```
uses calwin32

  NWFSE_LSL_LOGICAL_BOARD_STATS = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved0 : nuint16;
    padding  : nuint16;
    LogTtlTxPackets : nuint32;
    LogTtlRxPackets : nuint32;
    LogUnclaimedPackets : nuint32;
    reserved1 : nuint32;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved0**

Is reserved (pass zero).

**LogTtlTxPackets**

Specifies the total number of packets transmitted.

**LogTtlRxPackets**

Specifies the total number of packets received.

**LogUnclaimedPackets**

Specifies the total number of unclaimed packets.

**reserved1**

Is reserved (pass zero).

# NWFSE_MEDIA_MGR_OBJ_INFO

Returns information about media manager objects. Used by NWGetMediaMgrObjInfo (page 103).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   FSE_MM_OBJ_INFO            fseMMObjInfo ;
} NWFSE_MEDIA_MGR_OBJ_INFO;
```

## Delphi Structure

```
uses calwin32

 NWFSE_MEDIA_MGR_OBJ_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    fseMMObjInfo : FSE_MM_OBJ_INFO;
  End;
```

## Fields

**serverTimeAndVconsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**fseMMObjInfo**

Pointer to FSE_MM_OBJ_INFO (page 198).

# NWFSE_MEDIA_MGR_OBJ_LIST

Returns the media manager object list and the media manager object children's list. Used by NWGetMediaMgrObjList (page 105).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    nextStartObjNum ;
   nuint32                    objCount ;
   nuint32                    objs [FSE_MAX_OBJECTS];
} NWFSE_MEDIA_MGR_OBJ_LIST;
```

## Delphi Structure

```
uses calwin32

  NWFSE_MEDIA_MGR_OBJ_LIST = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    nextStartObjNum : nuint32;
    objCount : nuint32;
    objs : Array[0.. FSE_MAX_OBJECTS -1] Of nuint32;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**nextStartObjNum**

Contains the number to be passed as the `startNumber` parameter on the next call. When this field is -1, all information has been processed.

**objCount**

Specifies the number of object IDs returned.

**objs**

Specifies the list of object IDs.

# NWFSE_MEDIA_NAME_LIST

Returns the media name by using a media number. Used by NWGetMediaNameByMediaNum (page 108).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
} NWFSE_MEDIA_NAME_LIST;
```

## Delphi Structure

```
uses calwin32

 NWFSE_MEDIA_NAME_LIST = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

# NWFSE_MLID_BOARD_INFO

Contains a list of each Multiple Link Interface Driver (MLID) on a specified server. Used by NWGetMLIDBoardInfo (page 110).

**Service:** Server Environment

**Defined In:** nwfse.h

## Syntax

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint8                     reserved;
   nuint8                     numberProtocols;
   MLID_BOARD_INFO            MLIDBoardInfo[FSE_MAX_NUM_BOARD_INFO];
} NWFSE_MLID_BOARD_INFO;
```

## Delphi Syntax

```
CONST FSE_MAX_NUM_BOARD_INFO = 18;

uses calwin32

  NWFSE_MLID_BOARD_INFO = packed Record
     serverTimeAndVConsoleInfo :SERVER_AND_VCONSOLE_INFO;
     reserved : nuint8;
     numberProtocols :nuint8;
     LIDBoardInfo : Array[0..(FSE_MAX_NUM_BOARD_INFO-1)] of
                    MLID_BOARD_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**numberProtocols**

Specifies the number of protocols bound to the specified board.

**MLIDBoardInfo**

Points to MLID_BOARD_INFO, which contains information about each MLID.

# NWFSE_NETWORK_ADDRESS

Returns the network address information. Used by NWGetServerConnInfo (page 143).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   addressType;
   nuint32   addressSize;
   pnuint8   address;
} NWFSE_NETWORK_ADDRESS;
```

## Delphi Syntax

```
Type
   NWFSE_NETWORK_ADDRESS = packed Record
      addressType : nuint32;
      addressSize : nuint32;
      address : pnuint8;
   End;
```

## Fields

**addressType**

Specifies the type of the network transport address:

1 IPX
2 IP
8 UDP
9 TCP

**addressSize**

Specifies the size (in bytes) of the buffer allocated for the address and the actual size of the returned address.

**address**

Specifies the physical address in binary form.

## Remarks

If the address is an IP address, four bytes of the address are in printable order.

---

**TIP:** The declaration of address types 1 and 2 is different from the standard declaration (found at Network Address Types).

---

# NWFSE_NETWORK_ROUTER_INFO

Returns information about a specified router on the network. Used by NWGetNetworkRouterInfo (page 116).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    NetIDNumber ;
   nuint16                    HopsToNet ;
   nuint16                    NetStatus ;
   nuint16                    TimeToNet ;
} NWFSE_NETWORK_ROUTER_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_NETWORK_ROUTER_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    NetIDNumber : nuint32;
    HopsToNet : nuint16;
    NetStatus : nuint16;
    TimeToNet : nuint16;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved (pass zero).

**NetIDNumber**

Specifies the network ID number used by the server.

**HopsToNet**

Specifies the number of routers to cross to get to the network.

**NetStatus**

Specifies the status of the network.

**`TimeToNet`**

Specifies the number of clock ticks to the network (roundtrip).

## Remarks

The `NetStatus` field can have the following values:

0x01 LOCALBIT
0x02 NETSTARTBIT
0x04 NETRELIABLEBIT
0x10 NETWANBIT

# NWFSE_NETWORK_ROUTERS_INFO

Returns information about the routers on a network. Used by NWGetNetworkRoutersInfo (page 118).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    NumberOfEntries ;
   ROUTERS_INFO               routersInfo [36];
} NWFSE_NETWORK_ROUTERS_INFO;
```

## Delphi Structure

```
uses calwin32

 NWFSE_NETWORK_ROUTERS_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    NumberOfEntries : nuint32;
    routersInfo : Array[0..35] Of ROUTERS_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**NumberOfEntries**

Contains the number of ROUTERS_INFO (page 305) structures in the buffer.

**routersInfo**

Contains the number of ROUTERS_INFO (page 305) structures in the buffer.

# NWFSE_NLM_INFO

Returns information about an NLM running on a server for the current connection. Used by NWGetNLMInfo (page 122).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   NLM_INFO                   NLMInfo ;
} NWFSE_NLM_INFO;
```

## Delphi Structure

```
uses calwin32

 NWFSE_NLM_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    NLMInfo : NLM_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**NLMInfo**

Contains an NLM_INFO (page 223) structure.

# NWFSE_NLM_LOADED_LIST

Returns a list of NLMs running on a server. Used by NWGetNLMLoadedList (page 124).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    numberNLMsLoaded ;
   nuint32                    NLMsInList ;
   nuint32                    NLMNums [FSE_NLM_NUMS_RETURNED_MAX];
} NWFSE_NLM_LOADED_LIST;
```

## Delphi Structure

```
uses calwin32

NWFSE_NLM_LOADED_LIST = packed Record
  serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
  reserved : nuint16;
  padding  : nuint16;
  numberNLMsLoaded : nuint32;
  NLMsInList : nuint32;
  NLMNums : Array[0..FSE_NLM_NUMS_RETURNED_MAX-1] Of nuint32;
End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**numberNLMsLoaded**

Specifies the total number of NLMs loaded on the server including hidden NLMs. No information will be returned about hidden NLMs.

**NLMsInList**

Specifies the number of valid NLM IDs returned in `NLMNums`. A valid NLM is an NLM whose information was placed in the buffer and does not include hidden NLMs.

**NLMNums**

A list containing the numbers assigned to NLMs loaded on the server.

# NWFSE_NLMS_RESOURCE_TAG_LIST

Returns the NLM's resource tag list. Used by NWGetNLMsResourceTagList (page 126).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    totalNumOfResourceTags ;
   nuint32                    packetResourceTags ;
   nuint8                     resourceTagBuf [512];
} NWFSE_NLMS_RESOURCE_TAG_LIST;
```

## Delphi Structure

```
uses calwin32

  NWFSE_NLMS_RESOURCE_TAG_LIST = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    totalNumOfResourceTags : nuint32;
    packetResourceTags : nuint32;
    resourceTagBuf : Array[0..511] Of nuint8;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**totalNumOfResourceTags**

Specifies the total number of resource tags the NLM is using.

**packetResourceTags**

Specifies the number of resource tags the structure contains.

**resourceTagBuf**

Contains the resourceTagBuf (page 304) structure.

# NWFSE_OS_VERSION_INFO

Returns operating system version information. Used by NWGetOSVersionInfo (page 128).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint8                     OSMajorVersion ;
   nuint8                     OSMinorVersion ;
   nuint8                     OSRevisionNum ;
   nuint8                     accountingVersion ;
   nuint8                     VAPVersion ;
   nuint8                     queueingVersion ;
   nuint8                     securityRestrictionsLevel ;
   nuint8                     bridgingSupport ;
   nuint32                    maxNumOfVolumes ;
   nuint32                    numOfConnSlots ;
   nuint32                    maxLoggedInConns ;
   nuint32                    maxNumOfNameSpaces ;
   nuint32                    maxNumOfLans ;
   nuint32                    maxNumOfMediaTypes ;
   nuint32                    maxNumOfProtocols ;
   nuint32                    maxMaxSubdirTreeDepth ;
   nuint32                    maxNumOfDataStreams ;
   nuint32                    maxNumOfSpoolPrinters ;
   nuint32                    serialNum ;
   nuint16                    applicationNum ;
} NWFSE_OS_VERSION_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_OS_VERSION_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    OSMajorVersion : nuint8;
    OSMinorVersion : nuint8;
    OSRevisionNum : nuint8;
    accountingVersion : nuint8;
    VAPVersion : nuint8;
    queueingVersion : nuint8;
    securityRestrictionsLevel : nuint8;
    bridgingSupport : nuint8;
    maxNumOfVolumes : nuint32;
```

```
    numOfConnSlots : nuint32;
    maxLoggedInConns : nuint32;
    maxNumOfNameSpaces : nuint32;
    maxNumOfLans : nuint32;
    maxNumOfMediaTypes : nuint32;
    maxNumOfProtocols : nuint32;
    maxMaxSubdirTreeDepth : nuint32;
    maxNumOfDataStreams : nuint32;
    maxNumOfSpoolPrinters : nuint32;
    serialNum : nuint32;
    applicationNum : nuint16;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Reserved.

**OSMajorVersion**

Specifies the major version number of the OS.

**OSMinorVersion**

Specifies the minor version number of the OS.

**OSRevisionNum**

Specifies the version revision letter of the OS.

**accountingVersion**

Specifies the version of the accounting subsystem.

**VAPVersion**

Is not used.

**queueingVersion**

Specifies the queueing version number.

**securityRestrictionsLevel**

Specifies the security restriction version number.

**bridgingSupport**

Specifies the internet bridge support version number.

**maxNumOfVolumes**

Contains the maximum number of volumes that can be simultaneously mounted on the server.

**numOfConnSlots**

Specifies the maximum number of connections that can be used simultaneously on the server.

**maxLoggedInConns**

Contains the maximum number of connections that can be used simultaneously on the server.

**maxNumOfNameSpaces**

Specifies the maximum number of name spaces that can be simultaneously loaded on the server.

**maxNumOfLans**

Specifies the maximum number of LAN boards that can be used on the server.

**maxNumOfMediaTypes**

Specifies the maximum number of different media types allowed on the server.

**maxNumOfProtocols**

Specifies the maximum number of protocol stacks that can be used on the server.

**maxMaxSubdirTreeDepth**

Specifies the maximum depth of directories that can be used on the server.

**maxNumOfDataStreams**

Specifies the maximum number of data streams that can be used on the server.

**maxNumOfSpoolPrinters**

Specifies the maximum number of spool printers (default queue assignments) that can be used on the server.

**serialNum**

Specifies the serial number of the server.

**applicationNum**

Is included for backward compatibility.

# NWFSE_PACKET_BURST_INFO

Returns packet burst information. Used by NWGetPacketBurstInfo (page 130).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   PACKET_BURST_INFO          packetBurstInfo ;
} NWFSE_PACKET_BURST_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_PACKET_BURST_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    packetBurstInfo : PACKET_BURST_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved (pass zero).

**packetBurstInfo**

Specifies the PACKET_BURST_INFO structure containing information about packet bursts.

# NWFSE_PRINT_INFO

Returns Server Environment printing information. Used by NWGetServerConnInfo (page 143).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint8   printFlags;
   nuint8   tabSize
   nuint8   numberCopies;
   nuint8   printToFileFlag;
   nuint8   bannerFileName[14];
   nuint8   targetServerID;
   nuint8   formType;
} NWFSE_PRINT_INFO;
```

## Delphi Syntax

```
uses calwin32

  NWFSE_PRINT_INFO = packed RECORD
     printFlags : nuint8;
     tabSize : nuint8;
     numberCopies : nuint8;
     printToFileFlag : nuint8;
     bannerFileName : Array[1..14] of nuint8;
     targetServerID : nuint8;
     formType : nuint8;
  End;
```

## Fields

**`printFlags`**

Specifies the print flags:

| Value | Constant and Description |
|---|---|
| 0x08 | SFormSuppressBit specifies the print service suppresses automatic form feed after the print job is printed. |
| 0x10 | SCreateBit |
| 0x20 | SDeleteBit |
| 0x40 | STabBit |
| 0x80 | sBannerPageBit specifies the print service precedes the print job with a banner page. |

**tabSize**

Specifies the tab size which is a value between 1 and 18 inclusive (default setting is 0x08).

**numberCopies**

Specifies the number of copies (0 to 255) of the captured file that is printed (default setting is 0x0001). If 0x0000, nothing prints.

**printToFileFlag**

Specifies that the data is sent to a file rather than a printer.

**bannerFileName**

Specifies the name of the banner that is printed when a print job is submitted.

**targetServerID**

Specifies the server ID of the queue server servicing the job. If this field is set to 0xFFFFFFFF, any queue server can service the job. If the specified queue server is not attached to the queue, QMS removes the job from the queue.

**formType**

Specifies the type of form (0 to 255) a user must mount in the printer to print files captured to the LPT device. If the form currently mounted in the printer differs from the form type returned in this field, the NetWare server console displays a message instructing the console operator to mount the correct form. The default form is 0x0000.

## Remarks

The banner specified in `bannerFileName` will be printed only if the user has selected the option to print a banner. If a banner is not specified in `bannerFileName`, a default banner will print that contains the name of the file being printed.

# NWFSE_PROTOCOL_CUSTOM_INFO

Returns custom information about protocol stacks. Used by NWGetProtocolStackCustomInfo (page 135).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved0 ;
   nuint32                    customCount ;
   nuint8                     customStruct [512];
} NWFSE_PROTOCOL_CUSTOM_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_PROTOCOL_CUSTOM_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved0 : nuint16;
    padding  : nuint16;
    customCount : nuint32;
    customStruct : Array[0.. 512 -1] Of nuint8;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**customCount**

Contains the number of NWFSE_PROTOCOL_CUSTOM_INFO (page 278) structures in the buffer.

**customStruct**

Specifies the structure with the DWORD value of the custom counter, followed by a length preceded string describing the custom counter.

# NWFSE_PROTOCOL_ID_NUMS

Returns the protocol stack numbers using a media number or using a LAN board number. Used by NWGetProtocolStkNumsByMediaNum (page 141).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    stackIDCount ;
   nuint32                    stackIDs [FSE_STACK_IDS_MAX];
} NWFSE_PROTOCOL_ID_NUMS;
```

## Delphi Structure

```
uses calwin32

  NWFSE_PROTOCOL_ID_NUMS = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    stackIDCount : nuint32;
    stackIDs : Array[0.. FSE_STACK_IDS_MAX -1] Of nuint32;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**stackIDCount**

Specifies the number of protocol stack ID Number returned in `stackIDs` by NWGetProtocolStkNumsByMediaNum (page 141).

**stackIDs**

Specifies the list of stack ID numbers.

# NWFSE_PROTOCOL_STK_CONFIG_INFO

Returns information about the protocol stack configuration. Used by NWGetProtocolStackConfigInfo (page 133).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint8                     configMajorVersionNum ;
   nuint8                     configMinorVersionNum ;
   nuint8                     stackMajorVersionNum ;
   nuint8                     stackMinorVersionNum ;
   nuint8                     stackShortName [16];
} NWFSE_PROTOCOL_STK_CONFIG_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_PROTOCOL_STK_CONFIG_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    configMajorVersionNum : nuint8;
    configMinorVersionNum : nuint8;
    stackMajorVersionNum : nuint8;
    stackMinorVersionNum : nuint8;
    stackShortName : Array[0..15] Of nuint8;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**configMajorVersionNum**

Specifies the major version number of the configuration table.

**configMinorVersionNum**

Specifies the minor version number of the configuration table.

**stackMajorVersionNum**

Specifies the major version number of the protocol stack.

**stackMinorVersionNum**

Specifies the minor version number of the protocol stack.

**stackShortName**

Specifies the short protocol name; name used to register the stack with the LSL. The first byte is the length of the string followed by the string itself. It is not a null-terminated string.

# NWFSE_PROTOCOL_STK_STATS_INFO

Returns information about protocol stack statistics. Used by NWGetProtocolStackStatsInfo (page 137).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint8                     statMajorVersionNum ;
   nuint8                     statMinorVersionNum ;
   nuint16                    commonCounters ;
   nuint32                    validCountersMask ;
   nuint32                    totalTxPackets ;
   nuint32                    totalRxPackets ;
   nuint32                    ignoredRxPackets ;
   nuint16                    numCustomCounters ;
} NWFSE_PROTOCOL_STK_STATS_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_PROTOCOL_STK_STATS_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    statMajorVersionNum : nuint8;
    statMinorVersionNum : nuint8;
    commonCounters : nuint16;
    validCountersMask : nuint32;
    totalTxPackets : nuint32;
    totalRxPackets : nuint32;
    ignoredRxPackets : nuint32;
    numCustomCounters : nuint16;
  End;
```

## Fields

**`serverTimeAndVConsoleInfo`**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**`reserved`**

Is reserved for future use.

**statMajorVersionNum**

Specifies the major version number of the statistics table.

**statMinorVersionNum**

Specifies the minor version number of the statistics table.

**commonCounters**

Specifies the number of counters in the fixed portion of the table, current 3.

**validCountersMask**

Specifies which counters are valid, right most bit for the first counter; 0 = Counter is valid, 1 = Counter is invalid

**totalTxPackets**

Contains the total number of packets that were requested to be transmitted.

**totalRxPackets**

Specifies the total number of packets that were requested to be transmitted.

**ignoredRxPackets**

Specifies the number of incoming packets that were ignored by the stack.

**numCustomCounters**

Specifies the number of custom counters for the protocol stack.

# NWFSE_SERVER_INFO

Returns server information. Used by NWGetServerInfo (page 147).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint8                     serverAddress [12];
   nuint16                    hopsToServer ;
} NWFSE_SERVER_INFO;
```

## Delphi Structure

```
uses calwin32

NWFSE_SERVER_INFO = packed Record
   serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
   reserved : nuint16;
   padding  : nuint16;
   serverAddress : Array[0..11] Of nuint8;
   hopsToServer : nuint16;
End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**serverAddress**

contains the node address of the server.

**hopsToServer**

Contains the number of hops to the server.

# NWFSE_SERVER_SET_CATEGORIES

Returns information about server set categories. Used by NWGetServerSetCategories (page 149).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved ;
   nuint32                    numberOfSetCategories ;
   nuint32                    nextSequenceNumber ;
   nuint8                     categoryName [512];
} NWFSE_SERVER_SET_CATEGORIES;
```

## Delphi Structure

```
uses calwin32

  NWFSE_SERVER_SET_CATEGORIES = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    numberOfSetCategories : nuint32;
    nextSequenceNumber : nuint32;
    categoryName : Array[0..511] Of nuint8;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**numberOfSetCategories**

Specifies the total number of set categories supported on the server.

**nextSequenceNumber**

Specifies the number to be used for `startNum` on subsequent calls.

**categoryName**

Specifies the null-terminated (not length-preceded) string describing the category.

# NWFSE_SERVER_SET_CMDS_INFO

Returns server set command information. Used by NWGetServerSetCommandsInfo (page 151).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    numberOfSetCommands ;
   nuint32                    nextSequenceNumber ;
   nuint32                    setCmdType ;
   nuint32                    setCmdCategory ;
   nuint32                    setCmdFlags ;
   nuint8                     setNameAndValueInfo [500];
} NWFSE_SERVER_SET_CMDS_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_SERVER_SET_CMDS_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    numberOfSetCommands : nuint32;
    nextSequenceNumber : nuint32;
    setCmdType : nuint32;
    setCmdCategory : nuint32;
    setCmdFlags : nuint32;
    setNameAndValueInfo : Array[0..499] Of nuint8;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**numberOfSetCommands**

Specifies the total number of set commands for all the categories on the server.

**nextSequenceNumber**

Specifies the next number to be used for `startNum` on the next call.

**setCmdType**

Specifies how to interpret the command as follows:

```
O   FSE_TYPE_NUMBER
1   FSE_TYPE_BOOLEAN
2   FSE_TYPE_TICKS
3   FSE_TYPE_BLOCK_SHIFT (512*number)
4   FSE_TYPE_TIME_OFFSET ([+|-]hh:mm:ss converted to seconds )
5   FSE_TYPE_STRING
6   FSE_TYPE_TRIGGER
```

The following show the types of triggers:

```
0x00   FSE_TYPE_TRIGGER_OFF
0x01   FSE_TYPE_TRIGGER_ON
0x10   FSE_TYPE_TRIGGER_PENDING
0x20   FSE_TYPE_TRIGGER_SUCCESS
0x30   FSE_TYPE_TRIGGER_FAILED
```

**setCmdCategory**

Specifies the category the command belongs to, as follows:

```
0   FSE_COMMUNICATIONS
1   FSE_MEMORY
2   FSE_FILE_CACHE
3   FSE_DIR_CACHE
4   FSE_FILE_SYSTEM
5   FSE_LOCKS
6   FSE_TRANSACTION_TRACKING
7   FSE_DISK
8   FSE_TIME
9   FSE_NCP
10  FSE_MISCELLANEOUS
11  FSE_ERRORS
12  FSE_DIRECTORY_SERVICES
13  FSE_MULTIPROCESSOR
14  FSE_SERVICE_LOCATION_PROTOCOL
```

**setCmdFlags**

Specifies the ways in which this category may be changed, as follows:

```
0x01   FSE_STARTUP_ONLY
0x02   FSE_HIDE
0x04   FSE_ADVANCED
0x08   FSE_STARTUP_OR_LATER
0x10   FSE_NOT_SECURED_CONSOLE (Can't be performed on secured
                                console)
```

**setNameAndValueInfo**

Specifies the NULL-terminated string containing the name of the command at index zero, and the value that begins at strlen(name)+1. The string is not length-preceded.

# NWFSE_SERVER_SRC_INFO

Returns address information about servers known to a server with a given name. Used by NWGetServerSourcesInfo (page 153).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved ;
   nuint32                    numberOfEntries ;
   SERVERS_SRC_INFO           serverSrcInfo[42];
} NWFSE_SERVER_SRC_INFO;
```

## Delphi Structure

```
uses calwin32

 NWFSE_SERVER_SRC_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    numberOfEntries : nuint32;
    serversSrcInfo : Array[0..41] Of SERVERS_SRC_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**numberOfEntries**

Specifies the number of SERVERS_SRC_INFOs that were returned byNWGetServerSourcesInfo (page 153) .

**serverSrcInfo**

Specifies SERVERS_SRC_INFO (page 307).

# NWFSE_STATS_INFO

Returns Server Environment statistical information. Used by NWGetServerConnInfo (page 143).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint8    totalBytesRead[6];
   nuint8    totalBytesWritten[6];
   nuint32   totalRequests;
} NWFSE_STATS_INFO;
```

## Delphi Syntax

```
uses calwin32

  NWFSE_STATS_INFO = packed RECORD
      totalBytesRead : Array[1..6] of nuint8;
      totalBytesWritten : Array[1..6] of nuint8;
      totalRequests : nuint32;
  End;
```

## Fields

**`totalBytesRead`**

Specifies the number of bytes the user read (48-bit value).

**`totalBytesWritten`**

Specifies the number of bytes the user wrote (48-bit value).

**`totalRequests`**

Specifies the number of requests the user sent.

# NWFSE_USER_INFO

Returns user information for a given connection. Used by NWGetUserInfo (page 155).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   USER_INFO                  userInfo ;
} NWFSE_USER_INFO;
```

## Delphi Structure

```
 uses calwin32

  NWFSE_USER_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding : nuint16;
    userInfo : USER_INFO;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**userInfo**

Contains a USER_INFO (page 312)structure for the connection. The UserInformation structure is defined in NWSERVST.H.

# NWFSE_VOLUME_INFO_BY_LEVEL

Returns volume information when the information level is specified. Used by NWGetVolumeInfoByLevel (page 157).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    infoLevel ;
   VOLUME_INFO_BY_LEVEL       volumeInfo ;
} NWFSE_VOLUME_INFO_BY_LEVEL;
```

## Delphi Structure

```
uses calwin32

  NWFSE_VOLUME_INFO_BY_LEVEL = packed Record
    serverAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    infoLevel : nuint32;
    volumeInfo : VOLUME_INFO_BY_LEVEL;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved (pass zero).

**infoLevel**

Specifies the information level to be returned.

If this field is set to 1, the `volInfoDef` field of the VOLUME_INFO_BY_LEVEL (page 319) structure will be used.

If this field is set to 2, the `volInfoDef2` field of the VOLUME_INFO_BY_LEVEL structure will be used.

**volumeInfo**

Points to the VOLUME_INFO_BY_LEVEL (page 319) structure containing the specified volume information.

# NWFSE_VOLUME_SEGMENT_LIST

Returns the volume segment list. Used by NWGetVolumeSegmentList (page 159).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo ;
   nuint16                    reserved ;
   nuint32                    numOfVolumeSegments ;
   VOLUME_SEGMENT             volumeSegment [42];
} NWFSE_VOLUME_SEGMENT_LIST;
```

## Delphi Structure

```
uses calwin32

  NWFSE_VOLUME_SEGMENT_LIST = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    numOfVolumeSegments : nuint32;
    volumeSegment : Array[0..41] Of VOLUME_SEGMENT;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved for future use.

**numOfVolumeSegments**

Specifies the number of volume segments on the volume.

**volumeSegment**

Specifies the volume information structures for all the volume segments on the volume. Only the number of structures will contain valid data.

# NWFSE_VOLUME_SWITCH_INFO

Returns information about volume switches. Used by NWGetVolumeSwitchInfo (page 161).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO    serverTimeAndVConsoleInfo ;
   nuint16                     reserved ;
   nuint32                     totalLFSCounters ;
   nuint32                     CurrentLFSCounters ;
   nuint32                     LFSCounters [128];
} NWFSE_VOLUME_SWITCH_INFO;
```

## Delphi Structure

```
uses calwin32

  NWFSE_VOLUME_SWITCH_INFO = packed Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    padding  : nuint16;
    totalLFSCounters : nuint32;
    CurrentLFSCounters : nuint32;
    LFSCounters : Array[0..127] Of nuint32;
  End;
```

## Fields

**serverTimeAndVConsoleInfo**

Specifies the SERVER_AND_VCONSOLE_INFO (page 306) structure containing the time since the server was brought up. This time is returned in ticks (approximately 1/8 of a second). When this parameter reaches 0xFFFFFFFF, it wraps to zero.

**reserved**

Is reserved (pass zero).

**totalLFSCounters**

Specifies the total number of FS counters.

**CurrentLFSCounters**

Specifies the current number of FS counters.

**LFSCounters**

Specifies the number of LFS counters.

# NW_GUID

Contains the server's Global Universal Identification (GUID).

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint8   GUID[16];
} NW_GUID;
```

## Delphi Structure

```
Type
    NW_GUID = Array[0..15] of nuint8;
```

## Fields

**GUID**

Specifies the GUID for the server.

## Remarks

The server’s GUID is automatically generated by the OS the first time the server is brought up.

The Ethernet address of a LAN board is part the server's GUID. Once a LAN board is loaded, the GUID contains all zeros until a LAN board is loaded the first time.

The server GUID is kept in the registry so the initial GUID created for the server will be preserved across multiple reboots. If the registry is corrupted or deleted, the startup code will automatically generate a new server GUID as if this was the first time the server was brought up.

# PACKET_BURST_INFO

Returns packet burst information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   bigInvalidSlotCount ;
   nuint32   bigForgedPacketCount ;
   nuint32   bigInvalidPacketCount ;
   nuint32   bigStillTransmittingCount ;
   nuint32   stillDoingTheLastRequestCount ;
   nuint32   invalidCtrlRequestCount ;
   nuint32   ctrlInvalidMessageNumCount ;
   nuint32   ctrlBeingTornDownCount ;
   nuint32   bigRepeatTheFileReadCount ;
   nuint32   bigSendExtraCCCount ;
   nuint32   bigReturnAbortMessageCount ;
   nuint32   bigReadInvalidMessageNumCount ;
   nuint32   bigReadDoItOverCount ;
   nuint32   bigReadBeingTornDownCount ;
   nuint32   previousCtrlPacketCount ;
   nuint32   sendHoldOffMessageCount ;
   nuint32   bigReadNoDataAvailableCount ;
   nuint32   bigReadTryingToReadTooMuchCount ;
   nuint32   asyncReadErrorCount ;
   nuint32   bigReadPhysicalReadErrorCount ;
   nuint32   ctrlBadACKFragmentListCount ;
   nuint32   ctrlNoDataReadCount ;
   nuint32   writeDuplicateRequestCount ;
   nuint32   shouldntBeACKingHereCount ;
   nuint32   writeInconsistentPktLengthsCnt ;
   nuint32   firstPacketIsntAWriteCount ;
   nuint32   writeTrashedDuplicateRequestCnt ;
   nuint32   bigWriteInvalidMessageNumCount ;
   nuint32   bigWriteBeingTornDownCount ;
   nuint32   bigWriteBeingAbortedCount ;
   nuint32   zeroACKFragmentCountCount ;
   nuint32   writeCurrentlyTransmittingCount ;
   nuint32   tryingToWriteTooMuchCount ;
   nuint32   writeOutOfMemForCtrlNodesCount ;
   nuint32   writeDidntNeedThisFragmentCount ;
   nuint32   writeTooManyBuffsCheckedOutCnt ;
   nuint32   writeTimeOutCount ;
   nuint32   writeGotAnACKCount ;
   nuint32   writeGotAnACKCount1 ;
   nuint32   pollerAbortedTheConnCount ;
   nuint32   maybeHadOutOfOrderWritesCount ;
```

```
   nuint32   hadAnOutOfOrderWriteCount ;
   nuint32   movedTheACKBitDownCount ;
   nuint32   bumpedOutOfOrderWriteCount ;
   nuint32   pollerRemovedOldOutOfOrderCount ;
   nuint32   writeDidntNeedButRequestACKCnt ;
   nuint32   writeTrashedPacketCount ;
   nuint32   tooManyACKFragmentsCount ;
   nuint32   savedAnOutOfOrderPacketCount ;
   nuint32   connBeingAbortedCount ;
} PACKET_BURST_INFO;
```

## Delphi Structure

```
uses calwin32

  PACKET_BURST_INFO = packed Record
    bigInvalidSlotCount : nuint32;
    bigForgedPacketCount : nuint32;
    bigInvalidPacketCount : nuint32;
    bigStillTransmittingCount : nuint32;
    stillDoingTheLastRequestCount : nuint32;
    invalidCtrlRequestCount : nuint32;
    ctrlInvalidMessageNumCount : nuint32;
    ctrlBeingTornDownCount : nuint32;
    bigRepeatTheFileReadCount : nuint32;
    bigSendExtraCCCount : nuint32;
    bigReturnAbortMessageCount : nuint32;
    bigReadInvalidMessageNumCount : nuint32;
    bigReadDoItOverCount : nuint32;
    bigReadBeingTornDownCount : nuint32;
    previousCtrlPacketCount : nuint32;
    sendHoldOffMessageCount : nuint32;
    bigReadNoDataAvailableCount : nuint32;
    bigReadTryingToReadTooMuchCount : nuint32;
    asyncReadErrorCount : nuint32;
    bigReadPhysicalReadErrorCount : nuint32;
    ctrlBadACKFragmentListCount : nuint32;
    ctrlNoDataReadCount : nuint32;
    writeDuplicateRequestCount : nuint32;
    shouldntBeACKingHereCount : nuint32;
    writeInconsistentPktLengthsCnt : nuint32;
    firstPacketIsntAWriteCount : nuint32;
    writeTrashedDuplicateRequestCnt : nuint32;
    bigWriteInvalidMessageNumCount : nuint32;
    bigWriteBeingTornDownCount : nuint32;
    bigWriteBeingAbortedCount : nuint32;
    zeroACKFragmentCountCount : nuint32;
    writeCurrentlyTransmittingCount : nuint32;
    tryingToWriteTooMuchCount : nuint32;
    writeOutOfMemForCtrlNodesCount : nuint32;
    writeDidntNeedThisFragmentCount : nuint32;
    writeTooManyBuffsCheckedOutCnt : nuint32;
    writeTimeOutCount : nuint32;
```

```
    writeGotAnACKCount : nuint32;
    writeGotAnACKCount1 : nuint32;
    pollerAbortedTheConnCount : nuint32;
    maybeHadOutOfOrderWritesCount : nuint32;
    hadAnOutOfOrderWriteCount : nuint32;
    movedTheACKBitDownCount : nuint32;
    bumpedOutOfOrderWriteCount : nuint32;
    pollerRemovedOldOutOfOrderCount : nuint32;
    writeDidntNeedButRequestACKCnt : nuint32;
    writeTrashedPacketCount : nuint32;
    tooManyACKFragmentsCount : nuint32;
    savedAnOutOfOrderPacketCount : nuint32;
    connBeingAbortedCount : nuint32;
  End;
```

## Fields

**bigInvalidSlotCount**

Specifies the number of requests determined not to be packet burst requests. The requests are either not NCP connections, the connection number is greater than available connection slots, or they have an invalid connection structure.

**bigForgedPacketCount**

Specifies the number of times the station connection ID and unique IDs do not match the server'.

**bigInvalidPacketCount**

Specifies the number of times the request packet is determined to be invalid.

**bigStillTransmittingCount**

Specifies the number of times the previous request is still transmitting so the current requests are delayed.

**stillDoingTheLastRequestCount**

Specifies the number of times the previous request is still being processed and read.

**invalidCtrlRequestCount**

Specifies the number of times the type of the packet and the packet nature cannot be determined.

**ctrlInvalidMessageNumCount**

Specifies the number of packets received that do not have a message number corresponding to what is already been sent and what is being inquired about.

**ctrlBeingTornDownCount**

Specifies the number of times the connection is torn down.

**bigRepeatTheFileReadCount**

Specifies the number of times an old request had to be reread.

**bigSendExtraCCCount**

Specifies the number of times the completion code error needed to be resent.

**bigReturnAbortMessageCount**

Specifies the number of times an `Abort Message' was returned to the server.

**bigReadInvalidMessageNumCount**

Specifies the number of times the message number comparison between the station and server failed.

**bigReadDoItOverCount**

Specifies the number of times a request that had been processed was received again and reprocessed.

**bigReadBeingTornDownCount**

Specifies the number of times the status flag is set to abort so the read is torn down before being processed.

**previousCtrlPacketCount**

Specifies the number of times the server packet ID does not compare to the station's packet ID.

**sendHoldOffMessageCount**

Specifies the number of times the station's request

**bigReadNoDataAvailableCount**

Specifies the number of times nothing was able to be read from the request packet.

**bigReadTryingToReadTooMuchCount**

Specifies the number of times the request packet was greater than 64K so was unreadable.

**asyncReadErrorCount**

Specifies the number of times an error is returned trying to read the data in from the request packet.

**bigReadPhysicalReadErrorCount**

Specifies the number of times a physical read error was encountered while reading the request packet.

**ctrlBadACKFragmentListCount**

Specifies the number of times the fragments making up the packet burst exceeded the maximum fragments allowed.

**ctrlNoDataReadCount**

Specifies the number of times a control packet was received specifying to send more data after the request was thought to be over. The station error confirmed the error that no control data was read.

**writeDuplicateRequestCount**

Specifies the number of times a duplicate write was performed after a duplicate request was processed.

**shouldntBeACKingHereCount**

Specifies the number of times a request was being initialized when a fragment count was found indicating the station still needs to send some fragments.

**writeInconsistentPktLengthsCnt**

Specifies the number of times consistency checking on packet lengths failed.

**firstPacketIsntAWriteCount**

Specifies the number of times the first fragment packet is not a write request.

**writeTrashedDuplicateRequestCnt**

Specifies the number of times a duplicate write request was received without a need to acknowledge receipt of the request so it was trashed.

**bigWriteInvalidMessageNumCount**

Specifies the number of times the message number comparison between the station and the server failed.

**bigWriteBeingTornDownCount**

Specifies the number of times the write is aborted and the connection is torn down.

**bigWriteBeingAbortedCount**

Specifies the number of times the write is aborted and the station is cleared up.

**zeroACKFragmentCountCount**

Specifies the number of times the request needs to be retried so the acknowledge fragment count is zeroed out.

**writeCurrentlyTransmittingCount**

Specifies the number of times data was currently being transmitted so the transmission needed to complete before writing.

**tryingToWriteTooMuchCount**

Specifies the number of times an attempt was made to write a total length greater than 64K.

**writeOutOfMemForCtrlNodesCount**

Is currently unused.

**writeDidntNeedThisFragmentCount**

Specifies the number of times that everything needed is already acquired so the write process is ended.

**writeTooManyBuffsCheckedOutCnt**

Specifies the number of times the next packet was received without received the first packet. If the buffer cannot be kept, buffers are not released.

**writeTimeOutCount**

Specifies the number of times the write retry limit was exceeded.

**writeGotAnACKCount**

Specifies the number of times the client is resending part or all of the request back because the request timed out and there are missing pieces.

**writeGotAnACKCount1**

Specifies the number of times the client is resending part or all of the request back because the request timed out and there are missing pieces.

**pollerAbortedTheConnCount**

Specifies the number of times the poller lost track of the connection and the connection was cleared.

**maybeHadOutOfOrderWritesCount**

Specifies the number of times that extra packets for the write came in and needed to be reordered before a write completed.

**hadAnOutOfOrderWriteCount**

Specifies the number of times out of order packets were received and needed to be requested in the right order.

**movedTheACKBitDownCount**

Specifies the number of times out of order packets were reordered and information that was buffered up will not be re-requested by the out of order writes.

**bumpedOutOfOrderWriteCount**

Specifies the number of times packets needing to be reordered could not be reordered. The packets were thrown out and need to be resent.

**pollerRemovedOldOutOfOrderCount**

Specifies the number of times the write received buffers that were removed because they were on the out of order list too long.

**writeDidntNeedButRequestACKCnt**

Specifies the number of times the fragment count was zero indicating no new fragments were expected but the acknowledge bit was checked.

**writeTrashedPacketCount**

Specifies the number of times the cleaning up of additional receive buffers queued up during the execution of an error path.

**tooManyACKFragmentsCount**

Specifies the number of times a fragment placement needed to be adjusted.

**savedAnOutOfOrderPacketCount**

Specifies the number of times an out of order packet was received, saved for several seconds, and the buffer then moved onto the out of order write list.

**connBeingAbortedCount**

Specifies the number of times a station’s connection was aborted.

# PHYS_DSK_STATS

Returns physical disk statistics.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   systemElapsedTime ;
   nuint8    diskChannel ;
   nuint8    diskRemovable ;
   nuint8    driveType ;
   nuint8    controllerDriveNumber ;
   nuint8    controllerNumber ;
   nuint8    controllerType ;
   nuint32   driveSize ;/*in 4096 byte blocks*/
   nuint16   driveCylinders ;
   nuint8    driveHeads ;
   nuint8    sectorsPerTrack ;
   nuint8    driveDefinition [64];
   nuint16   IOErrorCount ;
   nuint32   hotFixStart ;
   nuint16   hotFixSize ;
   nuint16   hotFixBlockAvailable ;
   nuint8    hotFixDisabled ;
} PHYS_DSK_STATS;
```

## Delphi Structure

```
uses calwin32

PHYS_DSK_STATS = packed Record
    systemElapsedTime : nuint32;
    diskChannel : nuint8;
    diskRemovable : nuint8;
    driveType : nuint8;
    controllerDriveNumber : nuint8;
    controllerNumber : nuint8;
    controllerType : nuint8;
    driveSize : nuint32;
    driveCylinders : nuint16;
    driveHeads : nuint8;
    sectorsPerTrack : nuint8;
    driveDefinition : Array[0..63] Of nuint8;
    IOErrorCount : nuint16;
    hotFixStart : nuint32;
    hotFixSize : nuint16;
    hotFixBlockAvailable : nuint16;
```

```
    hotFixDisabled : nuint8;
  End;
```

## Fields

**systemElapsedTime**

Specifies how long the NetWare server has been up. This value is returned in units of approximately 1/18 second and is used to determine the amount of time that has elapsed between consecutive calls. When `systemElapsedTime` reaches 0xFFFFFFFF, it wraps back to zero.

**diskChannel**

Specifies the disk channel to which the disk unit is attached.

**diskRemovable**

Specifies whether a disk is removable (0 = nonremovable).

**driveType**

Specifies the type of drive, defined as follows:

```
1   XT
2   AT
3   SCSI
4   disk coprocessor
5   PS/2 with MFM Controller
6   PS/2 with ESDI Controller
7   Convergent Technology SBIC
50 to 255   Value-Added Disk Drive
```

**controllerDriveNumber**

Specifies the drive number of the disk unit relative to the controller number.

**controllerNumber**

Specifies the address on the physical disk channel of the disk controller.

**controllerType**

Specifies the number identifying the type (make and model) of the disk controller.

**driveSize**

Specifies the size of the physical drive in blocks (1 block = 4,096 bytes). The drive size does not include the portion of the disk reserved for Hot Fix redirection in the event of media errors.

**driveCylinders**

Specifies the number of physical cylinders on the drive.

**driveHeads**

Specifies the number of disk heads on the drive.

**sectorsPerTrack**

Specifies the number of sectors on each disk track (1 sector = 512 bytes).

**driveDefinition**

Specifies the make and model of the drive (NULL-terminated string).

**IOErrorCount**

Specifies the number of times I/O errors have occurred on the disk since the server was brought up.

**hotFixStart**

Specifies the first block of the disk Hot Fix Redirection Table. This field is meaningful only with SFT(tm) NetWare Level I or above. The redirection table is used to replace bad disk blocks with usable blocks in the event that a media failure occurs on the disk

**hotFixSize**

Specifies the total number of redirection blocks set aside on the disk for Hot Fix redirection. Some or all of these blocks may be in use. `hotFixSize` is meaningful only with SFT NetWare Level I or above. To determine the number of redirection blocks still available for future use, see `hotFixBlocksAvailable`

**hotFixBlockAvailable**

Specifies the number of redirection blocks that are still available. `hotFixBlockAvailable` is meaningful only on SFT NetWare Level I or above.

**hotFixDisabled**

Specifies whether Hot Fix is enabled or disabled. `hotFixDisabled` is meaningful only with SFT NetWare Level I or above (0 = enabled).

# resourceTagBuf

Returns information about resources used by NLMs on a server.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   number ;
   nuint32   signature ;
   nuint32   count ;
   nuint8    name [];
} resourceTagBuf;
```

## Fields

**number**

Contains the number assigned to the resource tag when it was allocated.

**signature**

Contains the signature of the resource tag.

**count**

Contains the number of this kind of tag that has been allocated.

**name**

Contains the name of the resource tag.

# ROUTERS_INFO

Returns information about the routers on a network.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint8    nodeAddress [6];
   nuint32   connectedLAN ;
   nuint16   routeHops ;
   nuint16   routeTime ;
} ROUTERS_INFO;
```

## Delphi Structure

```
uses calwin32

  ROUTERS_INFO = packed Record
    nodeAddress : Array[0..5] Of nuint8;
    connectedLAN : nuint32;
    routeHops : nuint16;
    routeTime : nuint16;
  End;
```

## Fields

**nodeAddress**

Contains the 6-byte network address of the router.

**connectedLAN**

Contains the LAN board number of the router.

**routeHops**

Contains the number of hops to the network specified by `networkNumber`.

**routeTime**

Contains the time (in ticks) to the network specified by `networkNumber`.

# SERVER_AND_VCONSOLE_INFO

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   currentServerTime ;
   nuint8    vconsoleVersion ;
   nuint8    vconsoleRevision ;
} SERVER_AND_VCONSOLE_INFO;
```

## Delphi Structure

```
uses calwin32

  SERVER_AND_VCONSOLE_INFO = packed Record
    currentServerTime : nuint32;
    vconsoleVersion : nuint8;
    vconsoleRevision : nuint8
  End;
```

## Fields

**`currentServerTime`**

Specifies the time in ticks (about 1/18 second) since the server was brought up. When `currentServerTime` reaches 0xFFFFFFFF, it wraps to 0.

---

**NOTE:** The tick count is derived from the original PC NTSC crystal frequency of 14.317180 MHz divided by 12 which is about 1.193MHz. The PC's 8253/8254 PIT further divides this clock by 65536 to generate an interrupt 18.207 times a second which approximates to one interrupt every 54.925 mS.

---

**`vconsoleVersion`**

Specifies the console version number and tracks the packet format.

**`vconsoleRevision`**

Specifies the console version revision number and tracks the packet format.

# SERVERS_SRC_INFO

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint8    serverNode [6];
   nuint32   connectedLAN ;
   nuint16   sourceHops ;
} SERVERS_SRC_INFO;
```

## Delphi Structure

```
uses calwin32

  SERVERS_SRC_INFO = packed Record
    serverNode : Array[0.. 6 -1] Of nuint8;
    connectedLAN : nuint32;
    sourceHops : nuint16;
  End;
```

## Fields

**serverNode**

Specifies the node address of the server.

**connectedLAN**

Specifies the LAN board number of the server.

**sourceHops**

Specifies the number of hops to the server.

# SPX_INFO

Returns information about the SPX protocol.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint16   SPXMaxConnsCount ;
   nuint16   SPXMaxUsedConns ;
   nuint16   SPXEstConnReq ;
   nuint16   SPXEstConnFail ;
   nuint16   SPXListenConnectReq ;
   nuint16   SPXListenConnectFail ;
   nuint32   SPXSendCount ;
   nuint32   SPXWindowChokeCount ;
   nuint16   SPXBadSendCount ;
   nuint16   SPXSendFailCount ;
   nuint16   SPXAbortedConn ;
   nuint32   SPXListenPacketCount ;
   nuint16   SPXBadListenCount ;
   nuint32   SPXIncomingPacketCount ;
   nuint16   SPXBadInPacketCount ;
   nuint16   SPXSuppressedPackCount ;
   nuint16   SPXNoSesListenECBCount ;
   nuint16   SPXWatchDogDestSesCount ;
} SPX_INFO;
```

## Delphi Structure

```
uses calwin32

 SPX_INFO = packed Record
    SPXMaxConnsCount : nuint16;
    SPXMaxUsedConns : nuint16;
    SPXEstConnReq : nuint16;
    SPXEstConnFail : nuint16;
    SPXListenConnectReq : nuint16;
    SPXListenConnectFail : nuint16;
    SPXSendCount : nuint32;
    SPXWindowChokeCount : nuint32;
    SPXBadSendCount : nuint16;
    SPXSendFailCount : nuint16;
    SPXAbortedConn : nuint16;
    SPXListenPacketCount : nuint32;
    SPXBadListenCount : nuint16;
    SPXIncomingPacketCount : nuint32;
    SPXBadInPacketCount : nuint16;
    SPXSuppressedPackCount : nuint16;
```

```
    SPXNoSesListenECBCount : nuint16;
    SPXWatchDogDestSesCount : nuint16;
  End;
```

## Fields

**SPXMaxConnsCount**

Specifies the maximum number of SPX™ connections allowed on the server.

**SPXMaxUsedConns**

Specifies the maximum number of SPX connections used at one time since the server was booted.

**SPXEstConnReq**

Specifies the total number of SPX connections established since the server was booted.

**SPXEstConnFail**

Specifies the number of times that an attempt to establish an SPX connection failed since the server was booted.

**SPXListenConnectReq**

Specifies the number of requests to post a listen since the server was booted.

**SPXListenConnectFail**

Specifies the number of times a request to post a listen failed since the server was booted.

**SPXSendCount**

Specifies the number of SPX packets sent since the server was booted.

**SPXWindowChokeCount**

Specifies the value used internally for debugging.

**SPXBadSendCount**

Specifies the number of bad packets sent since the server was booted.

**SPXSendFailCount**

Specifies the number of packets sent for which no acknowledgment was received since the server was booted.

**SPXAbortedConn**

Specifies the number of times a connection was aborted since the server was booted.

**SPXListenPacketCount**

Specifies the number of times a listen was posted on a socket since the server was booted.

**SPXBadListenCount**

Specifies the number of times a listen on a socket failed since the server was booted.

**SPXIncomingPacketCount**

Specifies the number of packets in the queue.

**SPXBadInPacketCount**

Specifies the number of bad SPX packets received since the server was booted.

**SPXSuppressedPackCount**

Specifies the number of times a duplicate SPX packet was received since the server was booted.

**SPXNoSesListenECBCount**

Specifies the number of times a listen was posted on a session that was not established since the server was booted.

**SPXWatchDogDestSesCount**

Specifies the number of times the watchdog destroyed a session since the server was booted.

# STACK_INFO

Returns information about the stack.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   StackNum ;
   nuint8    StackShortName [16];
} STACK_INFO;
```

## Delphi Structure

```
uses calwin32

  STACK_INFO = packed Record
    StackNum : nuint32;
    StackShortName : Array[0..15] Of nuint8;
  End;
```

## Fields

**`StackNum`**

Specifies the protocol number.

**`StackShortName`**

Specifies the protocol short name with `StackShortName` being the length, and the rest of the name following. It is not null-terminated.

# USER_INFO

Returns information about a user connection.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   connNum ;
   nuint32   useCount ;
   nuint8    connServiceType ;
   nuint8    loginTime [7];
   nuint32   status ;
   nuint32   expirationTime ;
   nuint32   objType ;
   nuint8    transactionFlag ;
   nuint8    logicalLockThreshold ;
   nuint8    recordLockThreshold ;
   nuint8    fileWriteFlags ;
   nuint8    fileWriteState ;
   nuint8    filler ;
   nuint16   fileLockCount ;
   nuint16   recordLockCount ;
   nuint8    totalBytesRead [6];
   nuint8    totalBytesWritten [6];
   nuint32   totalRequests ;
   nuint32   heldRequests ;
   nuint8    heldBytesRead [6];
   nuint8    heldBytesWritten [6];
} USER_INFO;
```

## Delphi Structure

```
uses calwin32

 USER_INFO = packed Record
    connNum : nuint32;
    useCount : nuint32;
    connServiceType : nuint8;
    loginTime : Array[0..6] Of nuint8;
    status : nuint32;
    expirationTime : nuint32;
    objType : nuint32;
    transactionFlag : nuint8;
    logicalLockThreshold : nuint8;
    recordLockThreshold : nuint8;
    fileWriteFlags : nuint8;
    fileWriteState : nuint8;
    filler : nuint8;
```

```
    fileLockCount : nuint16;
    recordLockCount : nuint16;
    totalBytesRead : Array[0..5] Of nuint8;
    totalBytesWritten : Array[0..5] Of nuint8;
    totalRequests : nuint32;
    heldRequests : nuint32;
    heldBytesRead : Array[0..5] Of nuint8;
    heldBytesWritten : Array[0..5] Of nuint8;
  End;
```

## Fields

**connNum**

Specifies the connection number of the user.

**useCount**

Specifies if the connection is in use:

1 Connection is in use
0 Connection is not in use

**connServiceType**

Specifies the connection type.

**loginTime**

Specifies the time the user logged in.

**status**

Specifies the status of the connection.

**expirationTime**

Specifies the expiration time.

**objType**

Specifies the object type of the user (usually 0x0100).

**transactionFlag**

Specifies the transaction tracking information.

**logicalLockThreshold**

Specifies the maximum number of logical locks a user can have.

**recordLockThreshold**

Specifies the maximum number of record locks the user can have.

**fileWriteFlags**

Specifies the writing status (includes active and stop bits):

1 FSE_WRITE
2 FSE_WRITE_ABORTED

**fileWriteState**

Specifies the writing status:

0 FSE_NOT_WRITING
1 FSE_WRITE_IN_PROGRESS
2 FSE_WRITE_BEING_STOPPED

**filler**

Is unused.

**fileLockCount**

Specifies the number of files the user locked.

**recordLockCount**

Specifies the number of records the user locked.

**totalBytesRead**

Specifies the number of bytes the user read (48-bit value).

**totalBytesWritten**

Specifies the number of bytes the user wrote (48-bit value).

**totalRequests**

Specifies the number of requests the user sent.

**heldRequests**

Specifies the number of requests held for accounting purposes.

**heldBytesRead**

Specifies the number of bytes the user read that have a hold on them for accounting purposes.

**heldBytesWritten**

Specifies the number of bytes the user wrote that have a hold on them for accounting purposes.

## Remarks

`connServiceType` can have the following values:

2 FSE_NCP_CONNECTION_TYPE
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

`status` can have the following values:

0x00000001 FSE_LOGGED_IN
0x00000002 FSE_BEING_ABORTED
0x00000004 FSE_AUDITED
0x00000008 FSE_NEEDS_SECURITY_CHANGE
0x00000010 FSE_MAC_STATION
0x00000020 FSE_AUTHENTICATED_TEMPORARY
0x00000040 FSE_AUDIT_CONNECTION_RECORDED
0x00000080 FSE_DSAUDIT_CONNECTION_RECORDED

# VERSION_INFO

Returns version information about the logical components of a NetWare server.

**Service:** Server Environment

**Defined In:** nwserver.h

## Structure

```
typedef struct
{
   nuint8    serverName [48];
   nuint8    fileServiceVersion ;
   nuint8    fileServiceSubVersion ;
   nuint16   maximumServiceConnections ;
   nuint16   connectionsInUse ;
   nuint16   maxNumberVolumes ;
   nuint8    revision ;
   nuint8    SFTLevel ;
   nuint8    TTSLevel ;
   nuint16   maxConnectionsEverUsed ;
   nuint8    accountVersion ;
   nuint8    VAPVersion ;
   nuint8    queueVersion ;
   nuint8    printVersion ;
   nuint8    virtualConsoleVersion ;
   nuint8    restrictionLevel ;
   nuint8    internetBridge ;
   nuint8    reserved [60];
} VERSION_INFO;
```

## Delphi Structure

```
uses calwin32

  VERSION_INFO = packed Record
    serverName : Array[0..47] Of nuint8;
    fileServiceVersion : nuint8;
    fileServiceSubVersion : nuint8;
    maximumServiceConnections : nuint16;
    connectionsInUse : nuint16;
    maxNumberVolumes : nuint16;
    revision : nuint8;
    SFTLevel : nuint8;
    TTSLevel : nuint8;
    maxConnectionsEverUsed : nuint16;
    accountVersion : nuint8;
    VAPVersion : nuint8;
    queueVersion : nuint8;
    printVersion : nuint8;
    virtualConsoleVersion : nuint8;
    restrictionLevel : nuint8;
```

```
    internetBridge : nuint8;
    reserved : Array[0..59] Of nuint8;
  End;
```

## Fields

**serverName**

Specifies the name of the queried server.

**fileServiceVersion**

Specifies the major NetWare version number of the running server.

**fileServiceSubVersion**

Specifies the minor NetWare version number of the running server.

**maximumServiceConnections**

Specifies the maximum number of connection slots (licensed or unlicensed) that the server has allocated since it came up, which is not the same as how many connections the server can support.

**connectionsInUse**

Specifies how many connections are currently using the server at the time of the query.

**maxNumberVolumes**

Specifies the maximum number of volumes the server supports.

**revision**

Specifies the revision level of the NetWare version number running on the server.

**SFTLevel**

Specifies which SFT level the server operating system is using.

**TTSLevel**

Specifies which TTS level the server operating system is using.

**maxConnectionsEverUsed**

Specifies the maximum number of licensed connections in use at one time since the server was installed.

**accountVersion**

Specifies the accounting version number.

**VAPVersion**

Specifies the VAP version number, which is only used with NetWare versions prior to 3.0.

**queueVersion**

Specifies the queuing version number.

**printVersion**

Specifies the Print Server version number.

**virtualConsoleVersion**

Specifies the Virtual Console version number.

**restrictionLevel**

Specifies the Security Restriction version number.

**internetBridge**

Specifies the Internet Bridge support version number.

**reserved**

Is reserved for future use.

## Remarks

To get product version information (major, minor, and revision), call the NWGetNetWareProductVersion (page 114) function.

# VOLUME_INFO_BY_LEVEL

Returns volume information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef union
{
   VOLUME_INFO_BY_LEVEL_DEF     volInfoDef ;
   VOLUME_INFO_BY_LEVEL_DEF2    volInfoDef2 ;
}  VOLUME_INFO_BY_LEVEL;
```

## Delphi Syntax

```
Type
  VOLUME_INFO_BY_LEVEL = packed RECORD
     case integer of
        1: (volinfoDef : VOLUME_INFO_BY_LEVEL_DEF);
        2: (volinfoDef2 : VOLUME_INFO_BY_LEVEL_DEF2);
  End;
```

## Fields

**volInfoDef**

Pointer to VOLUME_INFO_BY_LEVEL_DEF (page 320).

**volInfoDef2**

Pointer to VOLUME_INFO_BY_LEVEL_DEF2 (page 325).

# VOLUME_INFO_BY_LEVEL_DEF

Returns volumen information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   volumeType ;
   nuint32   statusFlagBits ;
   nuint32   sectorSize ;
   nuint32   sectorsPerCluster ;
   nuint32   volumeSizeInClusters ;
   nuint32   freedClusters ;
   nuint32   subAllocFreeableClusters ;
   nuint32   freeableLimboSectors ;
   nuint32   nonFreeableLimboSectors ;
   nuint32   nonFreeableAvailSubAllocSectors ;
   nuint32   notUsableSubAllocSectors ;
   nuint32   subAllocClusters ;
   nuint32   dataStreamsCount ;
   nuint32   limboDataStreamsCount ;
   nuint32   oldestDeletedFileAgeInTicks ;
   nuint32   compressedDataStreamsCount ;
   nuint32   compressedLimboDataStreamsCount ;
   nuint32   unCompressableDataStreamsCount ;
   nuint32   preCompressedSectors ;
   nuint32   compressedSectors ;
   nuint32   migratedFiles ;
   nuint32   migratedSectors ;
   nuint32   clustersUsedByFAT ;
   nuint32   clustersUsedByDirectories ;
   nuint32   clustersUsedbyExtendedDirs ;
   nuint32   totalDirectoryEntries ;
   nuint32   unUsedDirectoryEntries ;
   nuint32   totalExtendedDirectoryExtants ;
   nuint32   unUsedExtendedDirectoryExtants ;
   nuint32   extendedAttributesDefined ;
   nuint32   extendedAttributeExtantsUsed ;
   nuint32   directoryServicesObjectID ;
   nuint32   volumeLastModifiedDateAndTime ;
} VOLUME_INFO_BY_LEVEL_DEF;
```

## Delphi Structure

```
uses calwin32

  VOLUME_INFO_BY_LEVEL_DEF = packed Record
    volumeType : nuint32;
```

```
    statusFlagBits : nuint32;
    sectorSize : nuint32;
    sectorsPerCluster : nuint32;
    volumeSizeInClusters : nuint32;
    freedClusters : nuint32;
    subAllocFreeableClusters : nuint32;
    freeableLimboSectors : nuint32;
    nonFreeableLimboSectors : nuint32;
    nonFreeableAvailSubAllocSectors : nuint32;
    notUsableSubAllocSectors : nuint32;
    subAllocClusters : nuint32;
    dataStreamsCount : nuint32;
    limboDataStreamsCount : nuint32;
    oldestDeletedFileAgeInTicks : nuint32;
    compressedDataStreamsCount : nuint32;
    compressedLimboDataStreamsCount : nuint32;
    unCompressableDataStreamsCount : nuint32;
    preCompressedSectors : nuint32;
    compressedSectors : nuint32;
    migratedFiles : nuint32;
    migratedSectors : nuint2;
    clustersUsedByFAT : nunt32;
    clustersUsedByDirectories : nuint32;
    clustersUsedByExtendedDirs : nuint32;
    totalDirectoryEntries : nuint32;
    unUsedDirectoryEntries : nuint32;
    totalExtendedDirectoryExtants : nuint32;
    unUsedExtendedDirectoryExtants : nuint32;
    extendedAttributesDefined : nuint32;
    extendedAttributeExtantsUsed : nuint32;
    directoryServicesObjectID : nuint32;
    volumeLastModifiedDateAndTime : nuint32;
  End;
```

## Fields

**volumeType**

Specifies the defined type of the current volume.

**statusFlagBits**

Specifies the options that are currently available on the volume.

**sectorSize**

Specifies the sector size (in bytes).

**sectorsPerCluster**

Specifies the number of sectors per cluster or block.

**volumeSizeInClusters**

Specifies the size of the volume (in clusters or blocks).

**freedClusters**

Specifies the number of clusters or blocks that are currently free for allocation (does not include space that is currently available from deleted or limbo files, nor space that could be reclaimed from the suballocation file system).

**subAllocFreeableClusters**

Specifies the space that can be reclaimed from the suballocation file system.

**freeableLimboSectors**

Specifies the disk space (in clusters or blocks) that can be freed from deleted files.

**nonFreeableLimboSectors**

Specifies the disk space (in clusters or blocks) that is currently in deleted files and is not aged enough to be classified as `freeableLimboClusters`.

**nonFreeableAvailSubAllocSectors**

Specifies the space available to the suballocation file system, but not freeable to return as clusters or blocks.

**notUsableSubAllocSectors**

Specifies the disk space that is wasted by the suballocation file system. These clusters cannot be allocated by the suballocation system or used a regular clusters or blocks.

**subAllocClusters**

Specifies the disk space being used by the suballocation file system.

**dataStreamsCount**

Specifies the number of data streams for real files that have data allocated to them.

**limboDataStreamsCount**

Specifies the number of data streams for deleted files that have data allocated to them.

**oldestDeletedFileAgeInTicks**

Specifies the current age of the oldest file (in ticks).

**compressedDataStreamsCount**

Specifies the number of data streams for compressed real files.

**compressedLimboDataStreamsCount**

Specifies the number of data streams for compressed deleted files.

**unCompressableDataStreamsCount**

Specifies the number of data streams that are not compressable (real and deleted).

**preCompressedSectors**

Specifies the amount of disk space that was allocated to all files before they were compressed (includes "hole" space).

**compressedSectors**

Specifies the amount of disk space that is used by all compressed files.

**migratedFiles**

Specifies the number of migrated files.

**migratedSectors**

Specifies the amount of migrated disk space (in sectors).

**clustersUsedByFAT**

Specifies the amount of disk space (in clusters or blocks) being used by the FAT table.

**clustersUsedByDirectories**

Specifies the amount of disk space (in clusters or blocks) being used by directories.

**clustersUsedbyExtendedDirs**

Specifies the amount of disk space (in clusters or blocks) being used by the extended directory space.

**totalDirectoryEntries**

Specifies the total number of directories that are available on the volume.

**unUsedDirectoryEntries**

Specifies the total number of directory entries that are not in use on the volume.

**totalExtendedDirectoryExtants**

Specifies the amount of extended directory space extants (128 bytes each) that are available on the volume.

**unUsedExtendedDirectoryExtants**

Specifies the amount of extended directory space extants (128 bytes each) that are not in use on the volume.

**extendedAttributesDefined**

Specifies the number of extended attributes that are defined on the volume.

**extendedAttributeExtantsUsed**

Specifies the number of extended directory extants that are used by the extended attributes.

**directoryServicesObjectID**

Specifies the NDS ID for the volume.

**volumeLastModifiedDateAndTime**

Specifies the last time any file or subdirectory on the volume was modified (as tracked by the OS).

## Remarks

`volumeType` can have the following values:

0 VINetWare386
1 VINetWare286
2 VINetWare386x30
3 VINetWare386v31

`statusFlagBits` can have the following values:

0x01 SubAllocEnableBit
0x02 CompressionEnabledBit
0x04 MigrationEnableBit
0x08 AuditingEnabledBit
0x10 ReadOnlyEnableBit
0x20 ImmediatePurgeBit
0x80000000 NSSVolumeBit

# VOLUME_INFO_BY_LEVEL_DEF2

Returns volume information.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   volumeActiveCount ;
   nuint32   volumeUseCount ;
   nuint32   mACRootIDs ;
   nuint32   volumeLastModifiedDateAndTime ;
   nuint32   volumeReferenceCount ;
   nuint32   compressionLowerLimit ;
   nuint32   outstandingIOs ;
   nuint32   outstandingCompressionIOs ;
   nuint32   compressionIOsLimit ;
} VOLUME_INFO_BY_LEVEL_DEF2;
```

## Delphi Structure

```
uses calwin32

 VOLUME_INFO_BY_LEVEL_DEF2 = packed Record
    volumeActiveCount : nuint32;
    volumeUseCount : nuint32;
    mACRootIDs : nuint32;
    volumeLastModifiedDateAndTime : nuint32;
    volumeReferenceCount : nuint32;
    compressionLowerLimit : nuint32;
    outstandingIOs : nuint32;
    outstandingCompressionIOs : nuint32;
    compressionIOsLimit : nuint32;
  End;
```

## Fields

**volumeActiveCount**

Specifies a per-volume count that indicates the volume currently has operations in progress that need to be completed before other actions (such as dismounting a volume) can be initiated.

**volumeUseCount**

Specifies a per-volume count that is incremented before directory cache requests and other writes to the directory can begin. This field is decremented when such operations complete.

**mACRootIDs**

Specifies the Directory Numbers of the Macintosh roots.

**volumeLastModifiedDateAndTime**

Specifies the last date and time that any subdirectory on the volume was modified.

**volumeReferenceCount**

Specifies the last modified reference count. This field is increment every time that MarkDirectoryChanged is called on the volume.

**compressionLowerLimit**

Specifies a value used to determine if a file will be smaller once it is compressed. On a volume with suballocation enabled, this value is set to 512; otherwise, it is determined (in part) by the defined block size of the volume.

**outstandingIOs**

Specifies the number of IO operations that are still in progress and have not yet completed.

**outstandingCompressionIOs**

Specifies the number of compression-related IO operations that are still in progress and have not yet completed.

**compressionIOsLimit**

Specifies a per-volume count of the number of IO operations that can be dedicated to compression as opposed to the number of regular reading and writing operations.

# VOLUME_SEGMENT

Returns information about the segments in a volume.

**Service:** Server Environment

**Defined In:** nwfse.h

## Structure

```
typedef struct
{
   nuint32   volumeSegmentDeviceNum ;
   nuint32   volumeSegmentOffset ;
   nuint32   volumeSegmentSize ;
} VOLUME_SEGMENT;
```

## Delphi Structure

```
uses calwin32

  VOLUME_SEGMENT = packed Record
    volumeSegmentDeviceNum : nuint32;
    volumeSegmentOffset : nuint32;
    volumeSegmentSize : nuint32;
  End;
```

## Fields

**volumeSegmentDeviceNum**

Specifies the device the segment is located on.

**volumeSegmentOffset**

Specifies the offset of the segment in bytes.

**volumeSegmentSize**

Specifies the segment size in bytes.

# Server Management Concepts

# 4

Server management allows you to manage server functions from the workstation. From the workstation you can

- Add a name space to a volume
- Mount and dismount volumes
- Excute an NCF fileson a server
- Load and unload NLMs
- Modifiy SET command values

# Server Management Tasks

5

This section describes the common tasks associated with Server Management.

## 5.1 Managing Volumes

1. To mount a volume using either a volume name or a volume number, call NWSMMountVolume (page 347).
2. To dismount a volume using a volume name, call NWSMDismountVolumeByName (page 336).
3. To dismount a volume using a volume number, call NWSMDismountVolumeByNumber (page 338).

## 5.2 Managing a Volume's Name Space

1. To add a specified name space to a volume on a server, call NWSMAddNSToVolume (page 334).

## 5.3 Managing NCF Files

1. To execute a selected NCF file on a specified server, call NWSMExecuteNCFFile (page 340).

## 5.4 Managing NLMs

1. To load a selected NLM on a server, call NWSMLoadNLM (page 342).
2. To unload a selected NLM on a server, call NWSMUnloadNLM (page 353).

## 5.5 Managing SET Values

1. To change a SET integer value, call NWSMSetDynamicCmdIntValue (page 349).
2. To change a SET string value, call NWSMSetDynamicCmdStrValue (page 351).

# Server Management Functions

6

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

# NWSMAddNSToVolume

Adds a specified name space to a mounted volume on a server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMAddNSToVolume
  (NWCONN_HANDLE    connHandle,
   nuint16          volNumber,
   nuint8           namspc);
```

## Delphi Syntax

```
uses calwin32

Function NWSMAddNSToVolume
  (connHandle : NWCONN_HANDLE;
   volNumber : nuint16;
   namspc : nuint8
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**volNumber**

(IN) Specifies the volume number on which the name space will be loaded.

**namspc**

(IN) Specifies the name space to load on the volume.

## Return Values

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

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x00BF | No add name space string given |

| | |
|---|---|
| 0x0205 | Unable to add specified name space to a volume |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x8998 | VOLUME_DOES_NOT_EXIST |
| 0x89FB | ERR_NCP_NOT_SUPPORTED |

## Remarks

You must be logged into `connHandle`, be permanently authenticated, and have console operator rights at the minimum to call NWSMAddNSToVolume.

`namspc` values cannot be ORed; they may be added on each call. `namspc` values follow:

NW_NS_MAC
NW_NS_NFS
NW_NS_FTAM
NW_NS_OS2

Execute ADD NAME SPACE only once for each non-DOS naming convention you want to store on a volume.

Before you can add a name space to a volume, the volume and the name space module must both be loaded. See the NWSMLoadNLM (page 342) function.

## NCP Calls

0x2222 135 5 Add Name Space To Volume

## See Also

NWSMLoadNLM (page 342)

# NWSMDismountVolumeByName

Dismounts a volume on a selected server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMDismountVolumeByName (
   NWCONN_HANDLE         connHandle,
   const nstr8 N_FAR    *volumeName);
```

## Delphi Syntax

```
uses calwin32

Function NWSMDismountVolumeByName
  (connHandle : NWCONN_HANDLE;
   volumeName : pnstr8
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**volumeName**

(IN) Points to the NULL-terminated name of the NetWare volume to dismount.

## Return Values

These are common return values; see Return Values (*NDK: Connection, Message, and NCP Extensions*)for more information.

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x00BF | Invalid `volumeName` string passed |
| 0x0204 | Unable to dismount specified volume name |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |

| | |
|---|---|
| 0x8998 | VOLUME_DOES_NOT_EXIST |
| 0x89FB | ERR_NCP_NOT_SUPPORTED |

## Remarks

You must be logged into `connHandle`, be permanently authenticated, and have console operator rights at the minimum to call NWSMDismountVolumeByName.

## NCP Calls

0x2222 131 4 Dismount Volume

## See Also

NWSMMountVolume (page 347)

# NWSMDismountVolumeByNumber

Dismounts a volume on a selected server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMDismountVolumeByNumber
  (NWCONN_HANDLE    connHandle,
   nuint16          volumeNumber);
```

## Delphi Syntax

```
uses calwin32

Function NWSMDismountVolumeByNumber
  (connHandle : NWCONN_HANDLE;
   volumeNumber : nuint16
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**volumeNumber**

(IN) Specifies the number of the NetWare volume to dismount.

## Return Values

These are common return values; see Return Values (*NDK: Connection, Message, and NCP Extensions*)for more information.

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x00BF | Invalid `volumeName` string passed |
| 0x0204 | Unable to dismount specified volume name |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |

| | |
|---|---|
| 0x8998 | VOLUME_DOES_NOT_EXIST |
| 0x89FB | ERR_NCP_NOT_SUPPORTED |

## Remarks

You must be logged into `connHandle`, be permanently authenticated, and have console operator rights at the minimum to call NWSMDismountVolumeByNumber.

## NCP Calls

0x2222 131 4 Dismount Volume

## See Also

NWSMMountVolume (page 347)

# NWSMExecuteNCFFile

Executes a selected NCF file on a specified server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMExecuteNCFFile (
   NWCONN_HANDLE         connHandle,
   const nstr8 N_FAR    *NCFFileName);
```

## Delphi Syntax

```
uses calwin32

Function NWSMExecuteNCFFile
  (connHandle : NWCONN_HANDLE;
   NCFFileName : pnstr8
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**NCFFileName**

(IN) Points to the NULL-terminated name of the file to execute.

## Return Values

These are common return values; see Return Values (*NDK: Connection, Message, and NCP Extensions*) for more information.

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x009E | No Path and Name field supplied |
| 0x0207 | Unable to execute NCF file |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |

| | |
|---|---|
| 0x8993 | INVALID_FILENAME |
| 0x89FB | ERR_NCP_NOT_SUPPORTED |

## Remarks

You must be logged into `connHandle`, be permanently authenticated, and have console operator rights at the minimum to call NWSMExecuteNCFFile.

`fileName` may include a volume and path in the following format:

```
{VOLUME NAME:}{PATH\...|file name}
```

## NCP Calls

0x2222 131 7 Execute NCF File

# NWSMLoadNLM

Loads a selected NLM on a server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMLoadNLM (
   NWCONN_HANDLE         connHandle,
   const nstr8 N_FAR    *loadCommand);
```

## Delphi Syntax

```
uses calwin32

Function NWSMLoadNLM
  (connHandle : NWCONN_HANDLE;
   loadCommand : pnstr8
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**loadCommand**

(IN) Points to the load command for the NLM. It must be NULL terminated; maximum length is 482 bytes.

## Return Values

These are common return values; see Return Values (*NDK: Connection, Message, and NCP Extensions*)for more information.

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x0202 | Unable to load specified module |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |

| | |
|---|---|
| 0x88FB | ERR_NCP_NOT_SUPPORTED |
| 0x899E | INVALID_FILENAME |

## Remarks

You must be logged in with the connection specified by `connHandle`, be permanently authenticated, and have supervisor rights to call NWSMLoadNLM.

`loadCommand` can be any string that is valid for the LOAD command on the console (except it does not include the actual "LOAD" command). It contains the NLM name (including the path if it is not in the server path), and command line parameters if any.

The LOAD command has the following format:

```
{VOLUME NAME:}{PATH\...}NLMname{.ext}{parameters}
```

NWSMLoadNLM does not support UNC paths.

The NLM does not have to include the extensions. .NLM will be assumed if the extensions are not included.

The server console LOAD command is documented in the NetWare server utilities documentation.

## NCP Calls

0x2222 131 1 Load A NLM

## See Also

NWSMUnloadNLM (page 353)

# NWSMLoadNLM2

Loads a selected NLM on a server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMLoadNLM2 (
   NWCONN_HANDLE     connHandle,
   const nstr8 N_FAR *loadCommand,
   pnuint32          loadNLMReturnCode);
```

## Delphi Syntax

```
uses calwin32

Function NWSMLoadNLM2
  (connHandle : NWCONN_HANDLE;
   loadCommand : const nstr8 N_FAR
   loadNLMReturnCode : pnuint32
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**loadCommand**

(IN) Points to the load command for the NLM. It must be NULL terminated; maximum length is 482 bytes.

**loadNLMReturnCode**

(OUT) Points to nuint32 variable which contains a return code from the loading NLM procedure in servers when the return code from NWSMLoadNLM2 is 0x0202. See the Remarks section for possible values.

## Return Values

These are common return values; see Return Values (*NDK: Connection, Message, and NCP Extensions*) for more information.

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x0202 | Unable to load specified module |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x88FB | ERR_NCP_NOT_SUPPORTED |
| 0x899E | INVALID_FILENAME |

## Remarks

You must be logged in with the connection specified by `connHandle`, be permanently authenticated, and have a minimum of console operator rights to call NWSMLoadNLM2.

`loadCommand` can be any string that is valid for the LOAD command on the console (except it does not include the actual "LOAD" command). It contains the NLM name (including the path if it is not in the server path), and command line parameters if any.

The LOAD command has the following format:

```
{VOLUME NAME:}{PATH\...}NLMname{.ext}{parameters}
```

NWSMLoadNLM2 does not support UNC paths.

NWSMLoadNLM2 is an enhancement version of NWSMLoadNLM. It provides extra information to explain why the function cannot load an NLM.

The following values are possibilities when the loadNLMReturnCode is 0x0202:

```
LOAD_COULD_NOT_FIND_FILE            1
LOAD_ERROR_READING_FILE             2
LOAD_NOT_NLM_FILE_FORMAT            3
LOAD_WRONG_NLM_FILE_VERSION         4
LOAD_REENTRANT_INITIALIZE_FAILURE   5
LOAD_CAN_NOT_LOAD_MULTIPLE_COPIES   6
LOAD_ALREADY_IN_PROGRESS            7
LOAD_NOT_ENOUGH_MEMORY              8
LOAD_INITIALIZE_FAILURE             9
LOAD_INCONSISTENT_FILE_FORMAT       10
LOAD_CAN_NOT_LOAD_AT_STARTUP        11
LOAD_AUTO_LOAD_MODULES_NOT_LOADED   12
LOAD_UNRESOLVED_EXTERNAL            13
LOAD_PUBLIC_ALREADY_DEFINED         14
LOAD_XDC_DATA_ERROR                 15
LOAD_NOT_OS_DOMAIN                 16
```

## NCP Calls

0x2222 131 1 Load A NLM

## See Also

NWSMLoadNLM (page 342), NWSMUnloadNLM (page 353)

# NWSMMountVolume

Mounts a volume on a selected server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMMountVolume (
   NWCONN_HANDLE         connHandle,
   const nstr8 N_FAR    *volumeName,
   pnuint32              volumeNumber);
```

## Delphi Syntax

```
uses calwin32

Function NWSMMountVolume
  (connHandle : NWCONN_HANDLE;
   volumeName : pnstr8;
   volumeNumber : pnuint32
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**volumeName**

(IN) Points to the name of the NetWare volume to mount (must be NULL terminated).

**volumeNumber**

(OUT) Points to the number of the volume.

## Return Values

These are common return values; see Return Values (*NDK: Connection, Message, and NCP Extensions*) for more information.

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

| | |
|---|---|
| 0x00BF | Invalid `volumeName` string passed |
| 0x0203 | VOLUME_ALREADY_MOUNTED |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89FB | ERR_NCP_NOT_SUPPORTED |
| 0x8998 | VOLUME_DOES_NOT_EXIST |

## Remarks

You must be logged into `connHandle`, be permanently authenticated, and have console operator rights at the minimum to call NWSMMountVolume.

If upon mounting a volume, an error occurs, and if the set parameters are such that VREPAIR will automatically execute, VREPAIR will execute and NWSMMountVolume will not return until VREPAIR has completed. The volume is then mounted.

## NCP Calls

0x2222 131 3 Mount Volume

## See Also

NWSMDismountVolumeByName (page 336), NWSMDismountVolumeByNumber (page 338)

# NWSMSetDynamicCmdIntValue

Changes the current value of a set command (that takes in integer values) on a specified server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMSetDynamicCmdIntValue (
   NWCONN_HANDLE         connHandle,
   const nstr8 N_FAR    *setCommandName,
   nuint32               cmdValue;
```

## Delphi Syntax

```
uses calwin32

Function NWSMSetDynamicCmdIntValue
  (connHandle : NWCONN_HANDLE;
   setCommandName : pnstr8;
   cmdValue : nuint32
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**setCommandName**

(IN) Points to the set parameter name. It must be NULL terminated.

**cmdValue**

(IN) Specifies the new value for the set command parameter.

## Return Values

These are common return values; see Return Values (*NDK: Connection, Message, and NCP Extensions*) for more information.

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

| | |
|---|---|
| 0x008C | Invalid type flag value |
| 0x00BF | No `setCommandName` string |
| 0x0206 | Unable to set the command |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89FB | ERR_NCP_NOT_SUPPORTED |

## Remarks

You must be logged into `connHandle`, be permanently authenticated, and have console operator rights at the minimum to call NWSMSetDynamicCmdIntValue.

The server console SET command is documented in the NetWare server utilities documentation. The SET values OFF/ON are treated as the integers 0 (zero) and 1 respectively; they are not treated as strings.

## NCP Calls

0x2222 131 6 Set Command Value

## See Also

NWGetServerSetCommandsInfo (page 151), NWGetServerSetCategories (page 149), NWSMSetDynamicCmdStrValue (page 351)

# NWSMSetDynamicCmdStrValue

Changes the current values of a set command (that takes in string values) on a specified server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMSetDynamicCmdStrValue (
   NWCONN_HANDLE         connHandle,
   const nstr8 N_FAR    *setCommandName,
   const nstr8 N_FAR    *cmdValue);
```

## Delphi Syntax

```
uses calwin32

Function NWSMSetDynamicCmdStrValue
  (connHandle : NWCONN_HANDLE;
   setCommandName : pnstr8;
   cmdValue : pnstr8
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**setCommandName**

(IN) Points to the parameter command name. It must be NULL terminated.

**cmdValue**

(IN) Points to the new value for the set command parameter. It must be NULL terminated.

## Return Values

These are common return values; see Return Values (*NDK: Connection, Message, and NCP Extensions*) for more information.

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

| | |
|---|---|
| 0x008C | Invalid type flag value |
| 0x00BF | No `setCommandName` string |
| 0x0206 | Unable to set the command |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89FB | ERR_NCP_NOT_SUPPORTED |

## Remarks

You must be logged into `connHandle`, be permanently authenticated, and have console operator rights at the minimum to call NWSMSetDynamicCmdStrValue.

The server console SET command is documented in the NetWare server utilities documentation. The SET values OFF/ON are treated as the integers 0 (zero) and 1 respectively; they are not treated as strings.

## NCP Calls

0x2222 131 6 Set Command Value

## See Also

NWGetServerSetCommandsInfo (page 151), NWGetServerSetCategories (page 149), NWSMSetDynamicCmdIntValue (page 349)

# NWSMUnloadNLM

Unloads a selected NLM on a server

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

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

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

**Service:** Server Management

## Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMUnloadNLM (
   NWCONN_HANDLE         connHandle,
   const nstr8 N_FAR    *NLMName);
```

## Delphi Syntax

```
uses calwin32

Function NWSMUnloadNLM
  (connHandle : NWCONN_HANDLE;
   NLMName : pnstr8
) : NWCCODE;
```

## Parameters

**connHandle**

(IN) Specifies the server connection handle which is being managed.

**NLMName**

(IN) Points to the name of the NLM to be unloaded (NULL-terminated with a maximum length of 482 bytes).

## Return Values

These are common return values; see Return Values (*NDK: Connection, Message, and NCP Extensions*) for more information.

| | |
|---|---|
| 0x0000 | SUCCESS |
| 0x009E | Bad file name or no file name given |
| 0x0202 | Unable to unload specified module |
| 0x8801 | INVALID_CONNECTION |

| | |
|---|---|
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89FB | ERR_NCP_NOT_SUPPORTED |
| 0x899E | INVALID_FILENAME |

## Remarks

You must be logged into `connHandle`, be permanently authenticated, and have supervisor rights to call NWSMUnloadNLM.

The `NLMName` parameter can be any string that is valid for the UNLOAD command on the console (except it does not include the actual "UNLOAD" command) and contains the NLM name.

The `NLMName` parameter has the following format:

```
NLMname{.ext}
```

NWSMUnloadNLM does not support UNC paths.

The NLM does not have to include the extensions. .NLM will be assumed if the extension is not included.

The server console UNLOAD command is documented in the NetWare server utilities documentation.

**NOTE:** NWSMUnloadNLM can return SUCCESS without unloading the specified NLM application if the NLM also has dependencies loaded. The return of SUCCESS signifies only that the server successfully processed the RPC.

## NCP Calls

0x2222 131 2 Unload A NLM

## See Also

NWSMLoadNLM (page 342)

# TTS Concepts

# 7

This section provides and overview of TTS services, its functions, and features.

## 7.1 TTS Introduction

The Transaction Tracking System™ (TTS™) software allows NetWare® servers to track transactions and ensure file integrity by backing out of or erasing interrupted or partially completed transactions. For example, the server can back out of a transaction if your application terminates unexpectedly while a transaction is in progress.

A transaction can include any series of requests that affect transactional files. TTS can monitor from 1 to 200 transactions at a time. The maximum number (50 to 200) is configurable.

TTS can track only one transaction at a time for each session. If a session sends several transactions to a server rapidly, TTS queues the transactions and services them one at a time. TTS lets you begin and end transactions, monitor TTS status, and access TTS information.

## 7.2 Implicit Transaction Tracking

Implicit transaction tracking requires no coding on your part. If TTS is installed and enabled on a NetWare® server, transaction tracking applies to any logical or physical record lock you place on a transactional file. If the workstation's connection is disrupted before the records are unlocked, TTS backs out of the transaction and restores the records to their original state.

For related information, see Section 7.3, "Explicit Transaction Tracking," on page 355.

## 7.3 Explicit Transaction Tracking

Explicit transaction tracking uses TTS. To signal an explicit transaction, bracket file update sequences with a pair of TTS functions.

- NWTTSBeginTransaction (page 369) signals the beginning of a transaction.
- NWTTSEndTransaction (page 371) signals the end of a transaction.

TTS tracks all transactional files accessed between these two functions and automatically places a physical lock on transactional files that you write to during this time. As with implicit tracking, if your connection is disrupted, TTS backs out of any modified files. You can force TTS to back out of the transaction by calling NWTTSAbortTransaction (page 367).

TTS affects only transactional files. To make a file transactional, set the file's transaction bit. This bit is a file attribute and can be modified using the File System service. (See Directory Entry Attributes in Volume Management. )

For related information, see Section 7.2, "Implicit Transaction Tracking," on page 355.

## 7.4 Transaction Tracking Process

A typical transaction scenario is a banking database application that writes a debit to one account, a credit to another account, and a note to a log. The dependencies among the three operations make it extremely important that they are performed as a single transaction.

If any one of the operations fails, the integrity of the other operations is undermined. TTS ensures data integrity under circumstances such as these. Below is a description of how TTS responds to a request to write to a transactional file.

1. TTS stores the new data in cache memory. The file itself remains unchanged.
2. TTS scans the target file on the server hard disk, finds the data to be changed (old data), and copies the old data to cache memory. TTS also records the name and directory path of the target file and the location and length of the old data (record) within the file. The target file on the server hard disk is still unchanged.
3. TTS writes the old data in cache memory to a transaction work file on the server hard disk. The transaction work file resides at the root level of a volume on the server. The file is flagged system and hidden. The target file on the server hard disk is still unchanged.
4. TTS finally writes the new data in cache memory to the target file on the server hard disk.

TTS repeats these steps for each write within a transaction. The transaction work file grows to accommodate the old data for each write. If the transaction is interrupted, TTS writes the contents of the transaction work file to the target file, thereby restoring the file to its pre-transaction state.

## 7.5 Implicit Tracking Threshold

TTS lets you read and modify the implicit tracking threshold. This is the number of logical and physical locks your application can set before implicit tracking takes effect. The default threshold is 0, meaning that whenever you place a logical or physical lock on a transactional file, the file will be tracked. A threshold of 0xFF means that implicit transactions for the associated lock type have been completed.

There are two reasons to modify the threshold value:

- If your application is performing explicit transactions but also locking records you don’t want to track, you can turn off implicit transactions.
- If your application always keeps one or more records locked, raising the threshold prevents these records from being tracked.

## 7.6 TTS Transaction Functions

These functions perform transaction tracking.

| Function | Header | Comment |
|---|---|---|
| NWTTSAbortTransaction | nwtts.h | Aborts all transactions, explicit or implicit. When this function returns successfully, all transactions have been successfully backed out of. |
| NWTTSBeginTransaction | nwtts.h | Begins an explicit transaction. |

| Function | Header | Comment |
|---|---|---|
| NWTTSEndTransaction | nwtts.h | Ends an explicit transaction and returns a transaction reference number. |

## 7.7 TTS Status and File Control Functions

These functions check and modify the status of TTS on a NetWare® server and access the TTS transaction bit flags associated with transactional files.

| Function | Header | Comment |
|---|---|---|
| NWDisableTTS | nwtts.h | Disables transaction tracking on a server. |
| NWEnableTTS | nwtts.h | Enables transaction tracking on a server. |
| NWTTSGetControlFlags | nwtts.h | Returns the transaction bits for files flagged as transactional. |
| NWTTSIsAvailable | nwtts.h | Verifies that the server supports transaction tracking. |
| NWTTSSetControlFlags | nwtts.h | Enables or disables automatic record locking when writing to transactional files. |
| NWTTSTransactionStatus | nwtts.h | Verifies whether a transaction has been written to disk. |

## 7.8 TTS Threshold Functions

These functions read and modify the connection and process thresholds affecting implicit transaction tracking.

| Function | Header | Comment |
|---|---|---|
| NWTTSGetConnectionThresholds | nwtts.h | Returns the number of logical and physical record locks allowed before implicit transactions begin. |
| NWTTSGetProcessThresholds | nwtts.h | Returns the number of explicit physical and logical record locks allowed before implicit locking begins. |
| NWTTSSetConnectionThresholds | nwtts.h | Informs a server of how many explicit physical and logical record locks to permit before invoking implicit transactions. |
| NWTTSSetProcessThresholds | nwtts.h | Sets the number of logical and physical locks to perform before implicit locking begins. |

# TTS Tasks

8

This section describes *a task* associated with TTS services.

## 8.1 Enabling TTS

When TTS is disabled, NDS operations which require modifying the database on that server are also disabled.

1 At the console prompt of the server, type ENABLE TTS.

---

**NOTE:** If TTS was disabled because volume SYS: was full, log onto the server and delete unnecessary files from volume SYS:, then type ENABLE TTS at the console.

---

# TTS Functions

# 9

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

# NWDisableTTS

Disables transaction tracking on a NetWare® server

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 3.11, 3.12, 3.2

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

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

**Service:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWDisableTTS (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

Function NWDisableTTS
  (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 |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

## Remarks

Transaction Tracking is always enabled on 4.x servers due to NDS™ requirements; therefore, enabling or disabling transaction tracking is only supported on 3.x servers.

## NCP Calls

0x2222 23 207 Disable Transaction Tracking

# NWEnableTTS

Enables transaction tracking on a NetWare server

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 3.11, 3.12, 3.2

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

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

**Service:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWEnableTTS  (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

Function NWEnableTTS
  (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 |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x8996 | SERVER_OUT_OF_MEMORY |
| 0x89C6 | NO_CONSOLE_PRIVILEGES |

## Remarks

Transaction Tracking is always enabled on 4.x servers due to NDS requirements; therefore, enabling or disabling transaction tracking is only supported on 3.x servers.

## NCP Calls

0x2222 23 208 Enable Transaction Tracking

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

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

# NWTTSAbortTransaction

Aborts all transactions, explicit and implicit

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSAbortTransaction  (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSAbortTransaction
  (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 |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89FE | DIRECTORY_LOCKED |
| 0x89FE | Transaction Restart |
| 0x89FF | LOCK_ERROR |

## Remarks

When NWTTSAbortTransaction is complete, all transactions will have been successfully backed out.

If a transaction is aborted, all Writes made since the beginning of a transaction are cancelled, and all files are returned to the state they were in before the transaction began.

NWTTSAbortTransaction releases the following record locks:

- Physical record locks generated by the NetWare server when an application tried to write an unlocked record.
- Physical or logical locks not released because of a file Write.

0x89FE indicates more than the threshold number of logical or physical records are still locked by the application. However, the transaction is finished and any locks being held are released. When this happens, the NetWare server automatically starts a new implicit transaction.

## NCP Calls

0x2222 34 3 TTS Abort Transaction

## See Also

NWTTSBeginTransaction (page 369), NWTTSEndTransaction (page 371)

# NWTTSBeginTransaction

Begins an explicit transaction

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSBeginTransaction  (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSBeginTransaction
  (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 |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |

## Remarks

NWTTSBeginTransaction tracks all transactional files currently open, and those opened during the transaction.

When data is written to a transaction file during a transaction, the NetWare server automatically generates a physical record lock for the region being written. If a lock already exists, no additional lock is generated. This automatic locking can be disabled by calling NWTTSSetControlFlags.

Any closing and unlocking of transaction files is delayed until either NWTTSEndTransaction or NWTTSAbortTransaction is executed. Logical and physical records are not unlocked until the end of the transaction if file writes are performed while the lock is in force.

## NCP Calls

0x2222 34 1 TTS Begin Transaction

## See Also

NWTTSAbortTransaction (page 367), NWTTSEndTransaction (page 371), NWTTSSetControlFlags (page 383)

# NWTTSEndTransaction

Ends an explicit transaction and returns a transaction reference number

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSEndTransaction  (
   NWCONN_HANDLE   conn,
   pnuint32        transactionNum);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSEndTransaction
  (conn : NWCONN_HANDLE;
   transactionNum : pnuint32
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**transactionNum**

(OUT) Points to the transaction reference number for the transaction being ended (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 |

| | |
|---|---|
| 0x89FE | Transaction Restart |
| 0x89FF | LOCK_ERROR |

## Remarks

The transaction number is used to verify a successful transaction completion to disk.

The transaction is not necessarily written to disk when the reference number is returned. A client must call NWTTSTransactionStatus to verify a transaction has been written to disk. If the NetWare server fails before all updates contained within the transaction have been written to disk, the transaction is backed out when the NetWare server is rebooted.

If transaction tracking is disabled, `transactionNum` can still determine when the transaction has been completely written to disk. Since `transactionNum` is optional, substitute NULL if no return values are desired.

NWTTSEndTransaction releases all physical record locks generated by the NetWare server when a Write is made to an unlocked record. In addition, physical or logical locks that were not released due to a file Write are unlocked at this time.

0x89FE indicates more than the threshold number of logical or physical records are still locked by the application. However, the transaction is finished and any locks being held are released. In this case, the NetWare server automatically starts a new implicit transaction.

## NCP Calls

0x2222 34 2 TTS End Transaction

## See Also

NWTTSAbortTransaction (page 367), NWTTSBeginTransaction (page 369), NWTTSTransactionStatus (page 387)

# NWTTSGetConnectionThresholds

Returns the number of logical and physical record locks allowed before implicit transactions begin

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSGetConnectionThresholds  (
   NWCONN_HANDLE   conn,
   pnuint8         logicalLockLevel,
   pnuint8         physicalLockLevel);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSGetConnectionThresholds
  (conn : NWCONN_HANDLE;
   logicalLockLevel : pnuint8;
   physicalLockLevel : pnuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**logicalLockLevel**

(OUT) Points to the number of logical record locks allowed before implicit transactions begin (0 to 255, optional).

**physicalLockLevel**

(OUT) Points to the number of physical record locks allowed before implicit transactions begin (0 to 255, 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 |

## Remarks

Both NWTTSSetConnectionThresholds and NWGetConnectionThreshold s are useful for applications changing the implicit application thresholds that later want to restore them. For example, NWTTSGetConnectionThresholds can get the number of logical and physical locks, and NWTTSSetConnectionThresholds can perform one of the following:

- Turn off implicit transactions. (Applications using only explicit transactions, but sometimes generating unnecessary implicit transactions, need to turn off all implicit transactions.)
- Set implicit thresholds for applications always keeping one or more records locked.

The default threshold for logical and physical locks is 0. 0xFF means implicit transactions for the lock type have been completed.

Both `physicalLockLevel` and `logicalLockLevel` are optional parameters. Substitute NULL if these parameters are not to be returned. However, all parameter positions must be filled.

## NCP Calls

0x2222 34 7 TTS Get Workstation Threshold

## See Also

NWTTSSetConnectionThresholds (page 381)

# NWTTSGetControlFlags

Returns the transaction bits for files flagged as transactional

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSGetControlFlags  (
   NWCONN_HANDLE   conn,
   pnuint8         controlFlags);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSGetControlFlags
  (conn : NWCONN_HANDLE;
   controlFlags : pnuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**controlFlags**

(OUT) Points to Transaction Tracking Control flags.

## 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

Transaction tracking control flags are only valid for files flagged as TTS. Bits 1 to 7 in `controlFlags` are reserved; bit 0 is defined below:

```
0x00   Automatic record locking is disabled
0x01   Automatic record locking is enabled
```

## NCP Calls

0x2222 34 9 TTS Get Transaction Bits

## See Also

NWTTSSetControlFlags (page 383)

# NWTTSGetProcessThresholds

Returns the number of explicit physical and logical record locks allowed before implicit locking begins

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSGetProcessThresholds  (
   NWCONN_HANDLE   conn,
   pnuint8         logicalLockLevel,
   pnuint8         physicalLockLevel);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSGetProcessThresholds
  (conn : NWCONN_HANDLE;
   logicalLockLevel : pnuint8;
   physicalLockLevel : pnuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**logicalLockLevel**

(OUT) Points to the number of explicit logical record locks allowed before implicit transactions begin (0 to 255, optional).

**physicalLockLevel**

(OUT) Points to the number of explicit physical record locks allowed before implicit transactions begin (0 to 255, 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 |

## Remarks

NWTTSGetProcessThresholds and NWTTSSetProcessThresholds are useful for applications changing the implicit process thresholds that later want to restore them. For example, NWTTSGetProcessThresholds can query an application for the number of logical and physical record locks allowed before an implicit transaction begins, and NWTTSSetProcessThresholds can perform one of the following:

- Turn off implicit transactions. (Applications intending to use only explicit transactions, but sometimes generate unnecessary implicit transactions, need to turn off all implicit transactions.)
- Set implicit thresholds for applications always keeping one or more records locked.

The default threshold for logical and physical locks is 0. 0xFF means no implicit transactions are allowed for the lock type.

Thresholds returned by NWTTSGetProcessThresholds are valid for the requesting application only. When the application terminates, the connection thresholds are restored.

Both `physicalLockLevel` and `logicalLockLevel` are optional parameters. Substitute NULL if these parameters are not to be returned. However, all parameter positions must be filled.

## NCP Calls

0x2222 34 5 TTS Get Application Threshold

## See Also

NWTTSSetProcessThresholds (page 385)

# NWTTSIsAvailable

Verifies the NetWare server supports transaction tracking

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSIsAvailable  (
   NWCONN_HANDLE   conn);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSIsAvailable
  (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 | Transaction Tracking is Unavailable |
| 0x8801 | INVALID_CONNECTION |
| 0x890A | NLM_INVALID_CONNECTION |
| 0x89FD | Transaction Tracking is Disabled |
| 0x89FF | Transaction Tracking is Available |

## Remarks

0x0000 does not indicate successful completion of NWTTSIsAvailable. Instead, 0x0000 indicates TTS is unavailable. The successful completion code is 0x89FF, TTS is available.

## NCP Calls

0x2222 34 0 TTS Is Available

# NWTTSSetConnectionThresholds

Sets the number of explicit physical and logical record locks to permit before invoking implicit transactions

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSSetConnectionThresholds  (
   NWCONN_HANDLE   conn,
   nuint8          logicalLockLevel,
   nuint8          physicalLockLevel);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSSetConnectionThresholds
  (conn : NWCONN_HANDLE;
   logicalLockLevel : nuint8;
   physicalLockLevel : nuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**logicalLockLevel**

(IN) Specifies the number of logical record locks to allow before implicit transactions begin (0 to 255).

**physicalLockLevel**

(IN) Specifies the number of physical record locks to allow before implicit transactions begin (0 to 255).

## 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

The return values are in effect for all applications, not just ones calling NWTTSSetConnectionThresholds.

The default threshold for logical and physical locks is 0. 0xFF means no implicit transactions for the lock type can be performed.

NWTTSSetConnectionThresholds and NWTTSGetConnectionThresholds are useful for applications changing the implicit application thresholds that later want to restore them.

For example, NWTTSGetConnectionThresholds can obtain the current number of logical and physical locks, and NWTTSSetConnectionThresholds can perform one of the following:

- Turn off implicit transactions. (Applications using only explicit transactions, but sometimes generate unnecessary implicit transactions, need to turn off all implicit transactions.)
- Set implicit thresholds for applications always keeping one or more records locked.

## NCP Calls

0x2222 34 8 TTS Set Workstation Thresholds

## See Also

NWTTSGetConnectionThresholds (page 373)

# NWTTSSetControlFlags

Enables or disables automatic record locking on Writes to transactional files

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSSetControlFlags  (
   NWCONN_HANDLE   conn,
   nuint8          controlFlags);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSSetControlFlags
  (conn : NWCONN_HANDLE;
   controlFlags : nuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**controlFlags**

(IN) Specifies the Transaction Tracking control flags.

## 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

NWTTSSetControlFlags allows a client to set the transaction bits in `controlFlag`.

Transaction tracking control flags are only valid for files flagged as transactional. Only bit 0 is used currently. Flag definitions follow:

```
0x00   Automatic record locking is disabled
0x01   Automatic record locking is enabled
```

## NCP Calls

0x2222 34 10 TTS Set Transaction Bits

# NWTTSSetProcessThresholds

Sets the number of logical and physical locks to perform before implicit locking begins

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API  NWTTSSetProcessThresholds  (
   NWCONN_HANDLE   conn,
   nuint8          logicalLockLevel,
   nuint8          physicalLockLevel);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSSetProcessThresholds
  (conn : NWCONN_HANDLE;
   logicalLockLevel : nuint8;
   physicalLockLevel : nuint8
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**logicalLockLevel**

(IN) Specifies the number of logical record locks to allow before implicit transactions begin (0-255).

**physicalLockLevel**

(IN) Specifies the number of physical record locks to allow before implicit transactions begin (0-255).

## 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 |

## Remarks

The thresholds set by NWTTSSetProcessThresholds are valid for the requesting application only. When the application terminates, the default workstation thresholds are restored.

NWTTSSetProcessThresholds either turns off implicit transactions or allows applications always keeping one or more records locked to work. Applications intending to use only explicit transactions, but sometimes generating unnecessary implicit transactions, can call NWTTSSetProcessThresholds to turn off all implicit transactions.

The default threshold for logical and physical locks is 0 unless the number has been changed by calling NWTTSSetConnectionThresholds. 0xFF means no implicit transactions for the lock type are performed.

## NCP Calls

0x2222 34 6 TTS Set Application Thresholds

## See Also

NWTTSGetProcessThresholds (page 377)

# NWTTSTransactionStatus

Verifies whether a transaction has been written to disk

**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:** Transaction Tracking System (TTS)

## Syntax

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

NWCCODE N_API NWTTSTransactionStatus (
   NWCONN_HANDLE   conn,
   nuint32         transactionNum);
```

## Delphi Syntax

```
uses calwin32

Function NWTTSTransactionStatus
  (conn : NWCONN_HANDLE;
   transactionNum : nuint32
) : NWCCODE;
```

## Parameters

**conn**

(IN) Specifies the NetWare server connection handle.

**transactionNum**

(IN) Specifies the transaction reference number (obtained from NWTTSEndTransaction).

## 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 | Transaction not written to disk |

## Remarks

NWTTSTransactionStatus can be called even if NWTTSEndTransaction returns TTS_DISABLED.

Applications should not wait for transactions to be written to disk unless it is absolutely necessary. Because the NetWare server caches algorithms, it may be up to five seconds before they are actually written. Transactions are written to disk in the order in which they terminate.

## NCP Calls

0x2222 34 4 TTS Transaction Status

## See Also

NWTTSEndTransaction (page 371)

# Server-Based Server Environment Concepts

# 10

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

Server Environment allows you to control various server operations and gather statistics about server operation. The functions can be divided into two categories: functions related to the FCONSOLE utility, and statistical information functions.

## 10.1 Prerequisites

A knowledge of underlying NetWare® subsystems may be helpful or necessary to interpret information returned by many of these functions. For example, interpreting connection task information requires a general knowledge of file system record locks and semaphores.

Many of these functions return restricted information, so they require Console Operator (or, in some cases, Supervisor) rights.

## 10.2 Potential Uses

The statistical functions return a wide variety of information that can be used in many ways, such as performance analysis and configuration reporting NLM applications. Among other possibilities, applications could identify system overloads, hardware failures, and potential bottlenecks.

## 10.3 Server-Based Server Environment Functions

The functions beginning with *SS* are all new functions available for NetWare 4.x which provide statistical information. The other functions are used for controlling the server and gathering statistical information. FCONSOLE provides a good example of the kinds of information provided by these functions.

The FCONSOLE server control functions allow you to

- Enable or disable transaction tracking
- Prohibit or allow users to log in
- Set the server time and date
- Broadcast messages to a group of workstations
- Clear a connection
- Down the server.
- Check whether a connection has console operator rights

The FCONSOLE functions also return many types of information, including

- NetWare version running on a server

- The remaining disk space for an object
- Semaphore information
- File usage and task information
- Disk cache statistics
- Disk utilization information
- LAN driver information
- Information about logical and physical records

The statistical information functions return many types of information, including

- Active connections for a server
- LAN and LSL information
- CPU information
- Cache information
- IPX and SPX information
- Information about known networks and servers
- File system information
- NLM information
- Packet Burst information
- User information
- Volume information

There is some overlap in the types of information returned by the FCONSOLE functions and the newer statistical information functions. The functions with an asterisk in the following table are new in NetWare 4.x.

**NOTE:** To safeguard the server from unauthorized tampering, most functions in this section require Console Operator rights and some require Supervisor rights.

| | |
|---|---|
| CheckConsolePrivileges | Determines whether the current connection has console operator rights |
| CheckNetWareVersion | Verifies compatibility of an application with the version of the NetWare® OS running on a server |
| ClearConnectionNumber | Clears a logical connection from a server |
| DisableFileServerLogin | Disables all logins to a server |
| DisableTransactionTracking | Disables transaction tracking on a server |
| DownFileServer | Brings the server down |
| EnableFileServerLogin | Enables logins to the server |
| EnableTransactionTracking | Enables transaction tracking on the server |
| GetBinderyObjectDiskSpaceLeft | Returns an objects remaining disk space |

| | |
|---|---|
| GetConnectionSemaphores* | Returns information about a connections open semaphores |
| GetConnectionsOpenFiles* | Returns information about a connections open files |
| GetConnectionsTaskInformation* | Returns information about a connections active tasks |
| GetConnectionsUsageStats* | Returns information about a connections usage |
| GetConnectionsUsingFile* | Returns information about connections using a given file |
| GetDiskCacheStats* | Returns information about disk caching on a server |
| GetDiskChannelStats* | Returns information about a servers disk channels |
| GetDriveMappingTable* | Returns information about drive mappings for a server |
| GetFileServerDateAndTime | Returns the date and time of the server |
| GetFileServerDescriptionStrings | Returns the name of the company that distributed this copy of NetWare |
| GetFileServerLANIOStats* | Returns information about packets sent and received by a server |
| GetFileServerLoginStatus | Determines whether logins are disabled or enabled on a server |
| GetFileServerMiscInformation* | Returns bindery and memory statistics for a server |
| GetFileServerName | Returns the name of a server |
| GetFileSystemStats* | Returns file system statistics for a server |
| GetLanDriverConfigInfo* | Returns configuration information for the LAN drivers installed on a server |
| GetLogicalRecordInformation* | Returns information about a logical record |
| GetLogicalRecordsByConnection* | Returns the logical records that a connection has logged with a server |
| GetPathFromDirectoryEntry* | Accesses a file path listed in a servers directory entry table (DET) |
| GetPhysicalDiskStats* | Returns information about a physical disk |
| GetPhysicalRecordLocksByFile* | Returns physical records that are locked in a file |
| GetPhysRecLockByConnectAndFile* | Returns a connections physical record locks within a file |
| GetSemaphoreInformation* | Returns information about a semaphore |
| GetServerInformation | Returns information about a server |

| | |
|---|---|
| SendConsoleBroadcast | Sends a message to a list of connections |
| SetFileServerDateAndTime | Sets the date and time of the server |
| SSGetActiveConnListByType* | Returns a list of active connections |
| SSGetActiveLANBoardList* | Returns information about the active LAN boards on a server |
| SSGetActiveProtocolStacks* | Returns protocol information for a server |
| SSGetCacheInfo* | Returns information about a servers cache buffers |
| SSGetCPUInfo* | Returns information about a servers CPU |
| SSGetDirCacheInfo* | Returns information about directory caching on a server |
| SSGetFileServerInfo* | Returns information about a server |
| SSGetFileSystemInfo* | Returns information about a servers file system |
| SSGetGarbageCollectionInfo* | Returns information about garbage collection on a server |
| SSGetIPXSPXInfo* | Returns information about IPX™/SPX™ use on a server |
| SSGetKnownNetworksInfo* | Returns information about the networks known to a server |
| SSGetKnownServersInfo* | Returns information about the servers known to a given server |
| SSGetLANCommonCounters* | Returns LAN common counters |
| SSGetLANConfiguration* | Returns information about LAN drivers on a server |
| SSGetLANCustomCounters* | Returns custom counters defined for a LAN driver |
| SSGetLoadedMediaNumberList* | Returns a list of media loaded on a server |
| SSGetLSLInfo* | Returns information about an LSL™ board |
| SSGetLSLLogicalBoardStats* | Returns information about LSL logical boards |
| SSGetMediaManagerObjChildList* | Returns a list of child objects for a given media manager object |
| SSGetMediaManagerObjInfo* | Returns media manager information |
| SSGetMediaManagerObjList* | Returns a list of media manager objects |
| SSGetMediaNameByNumber* | Returns a media name for a given media number |
| SSGetNetRouterInfo* | Returns information about network routing on a server |
| SSGetNetworkRoutersInfo* | Returns information about the routers on a network |

| | |
|---|---|
| SSGetNLMInfo* | Returns information about an NLM™ running on a server |
| SSGetNLMLoadedList* | Returns a list of NLM applications loaded on a server |
| SSGetNLMResourceTagList* | Returns information about resources used by NLM applications on a server |
| SSGetOSVersionInfo* | Returns information about the OS version running on a server |
| SSGetPacketBurstInfo* | Returns Packet Burst™ information for a server |
| SSGetProtocolConfiguration* | Returns configuration information about the protocols on a server |
| SSGetProtocolCustomInfo* | Returns custom information about a protocol stack |
| SSGetProtocolNumbersByLANBoard* | Returns a list of protocols for a given LAN board |
| SSGetProtocolNumbersByMedia* | Returns a list of protocols for a given media |
| SSGetProtocolStatistics* | Returns protocol statistics for a server |
| SSGetRouterAndSAPInfo* | Returns router and SAP information for a server |
| SSGetServerInfo* | Returns address and routing information for a server |
| SSGetServerSourcesInfo* | Returns server addresses known to a server with a given name |
| SSGetUserInfo* | Returns user information for a given connection |
| SSGetVolumeSegmentList* | Returns a list of volume segments for a given volume on a server |
| SSGetVolumeSwitchInfo* | Returns counts for volume access functions |

# Server-Based Server Environment Functions

# 11

This documentation alphabetically lists the server-based server environment functions and describes their purpose, syntax, parameters, and return values.

## 11.1 A*-GetD* Functions

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

# CheckConsolePrivileges

Determines whether the object on the current connection is a console operator (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWCheckConsolePrivileges (page 32))

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int CheckConsolePrivileges (void);
```

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

This function determines whether the object on the current connection has console operator rights. If the function returns 0, the object has console operator rights.

## See Also

Bindery Concepts

## CheckConsolePrivileges Example

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

main()
{
   int   ccode;
   printf ("\n\n");
   ccode = CheckConsolePrivileges ();
   if (ccode == 0)
      printf ("You HAVE console Operator rights.\n");
```

```
   else
      if (ccode == 198)
      printf ("You DO NOT have console Operator rights.\n");
}
```

# CheckNetWareVersion

Verifies compatibility between an application and the version of NetWare® running on the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWCCGetConnInfo)

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int CheckNetWareVersion (
   WORD   majorVersion,
   WORD   minorVersion,
   WORD   revisionNumber,
   WORD   minimumSFTlevel,
   WORD   minimumTTSlevel);
```

## Parameters

**majorVersion**

(IN) Specifies the minimum major version number that is compatible.

**minorVersion**

(IN) Specifies the minimum minor version number that is compatible.

**revisionNumber**

(IN) Specifies the minimum revision number that is compatible.

**minimumSFTlevel**

(IN) Specifies the minimum System Fault Tolerant™ ( SFT™) level that is compatible.

**minimumTTSlevel**

(IN) Specifies the minimum Transaction Tracking System™ ( TTS™) level that is compatible.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | COMPATIBLE |
| 1 | (0x01) | VERSION_NUMBER_TOO_LOW |

| Decimal | Hex | Constant |
|---|---|---|
| 2 | (0x02) | SFT_LEVEL_TOO_LOW |
| | | The version number is compatible, but the SFT level is too low. |
| 3 | (0x03) | TTS_LEVEL_TOO_LOW |
| | | The version number is compatible, but the TTS level is too low. |

## Remarks

Versions of NetWare are identified by major version number, minor version number, and (if applicable) revision number. The revision number is displayed as a letter (0=a, 1=b, and so on) to the user. For example, if a version number were 2.10a, 2 would be the major version number, 10 would be the minor version number, and 0 would be the revision number.

## CheckNetWareVersion Example

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

main()
{
   int    i;
   WORD   mv, msv, mr, msft, mtts;
   printf("min verSion, min subversion, min rev, min SFT, min TTS\n");
   scanf ("%d, %d, %d, %d, %d", &mv, &msv, &mr, &msft, &mtts);
   i = CheckNetWareVersion (mv, msv, mr, msft, mtts);
   printf ("return is %d\n",i);
}
```

# ClearConnectionNumber

Clears a logical connection from the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWClearConnectionNumber)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int ClearConnectionNumber (
   WORD   connectionNumber);
```

## Parameters

**connectionNumber**

(IN) Contains the identification number that the server assigns to a requesting workstation when the workstation attaches to the server.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

The first time a workstation attaches to a server, the server assigns it the first unused connection number in the File Server Connection Table. When a workstation detaches from a server, the server marks its connection number unused but reserves it in anticipation of reattachment. A reserved logical connection number is not reassigned to another workstation until it is the first unused connection number and the attaching workstation does not have a connection number reserved.

The server has room for 250 logical connections. The File Server Connection Table maintains the addresses of the workstations attached to the server.

This function blocks when successful.

This function closes a connection's open files and releases a connection's file locks. On a TTS server, this function causes a connection's transactions to be aborted.

When a requesting workstation clears its own connection, it can no longer communicate with the server.

The requesting workstation must have security equivalence to Supervisor rights.

## ClearConnectionNumber Example

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

main()
{
   int    ccode;
   WORD   connectionNumber;
   printf ("\n\n");
   printf ("Enter connection number to be cleared:  ");
   scanf ("%u", &connectionNumber);
   ccode = ClearConnectionNumber (connectionNumber);
   if (ccode == 198)
      printf ("\n\nNO CONSOLE RIGHTS\n");
   else if (ccode == 0)
      printf ("\n\nSUCCESSFULLY cleared connection number. %u\n",
               connectionNumber);
   else
      printf ("\nccode = %d\n", ccode);
}
```

# DisableFileServerLogin

Disables all logins to a server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWDisableFileServerLogin (page 35))

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int DisableFileServerLogin (void);
```

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

The DisableFileServerLogin function allows a console operator to restrict new accesses to the server during a crucial period of time such as before shutting down the server. (See also: DISABLE LOGIN console command.) If the workstation from which this call is made loses its connection to the server and no other logged objects have console Operator rights, logins can be reenabled by downing the server and rebooting, or by entering the ENABLE LOGIN command at the server console. The requesting workstation must have console Operator rights.

## See Also

EnableFileServerLogin (page 408), GetFileServerLoginStatus (page 442)

## DisableFileServerLogin Example

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

main()
{
   int   ccode;
```

```
   char  serverName[48];
   WORD  connID;
   printf ("\n\n");
   ccode = DisableFileServerLogin ();
   if (ccode == 198)
      printf ("Console Operator rights required for this
      utility\n");
   else if (ccode == 0)
   {
      connID = GetDefaultConnectionID ();
      GetFileServerName (connID, serverName);
      printf ("SUCCESSFUL\n\n");
      printf ("Disabled all new accesses to %s\n", serverName);
   }
   else
      printf ("Completion Code  =   %d\n", ccode);
}
```

# DisableTransactionTracking

Disables transaction tracking on the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWDisableTTS (page 362))

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 3.x

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int DisableTransactionTracking (void);
```

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

DisableTransactionTracking is usually used to disable transaction tracking temporarily. This function has no effect if TTS is not installed on the server.

Transaction tracking is also disabled by the server when the transaction volume is full.

The requesting workstation must have console Operator rights. (See also: DISABLE TTS console command.)

## See Also

EnableTransactionTracking (page 410)

## DisableTransactionTracking Example

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

main()
{
   int ccode;
```

```
   ccode = DisableTransactionTracking ();
   if (ccode == 0)
      printf ("Successfully disabled TRANSACTION TRACKING...\n");
   else
      printf ("ccode = %d\n", ccode);
}
```

# DownFileServer

Brings the server down (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWDownFileServer (page 37))

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int DownFileServer (
   int   forceFlag);
```

## Parameters

**`forceFlag`**

(IN) Indicates whether the server should be forced down even if files are open (0 = Server does not go down if there are open files).

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |
| 255 | (0xFF) | ERR_OPEN_FILES |

## Remarks

The requesting workstation must have security equivalence to Supervisor rights.

## DownFileServer Example

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <nwconio.h>
#include <nlm\nit\nwenvrn.h>
```

```
main()
{
   int   ccode;
   int   forceFlag;
   char  ans;
   printf ("\n\n");
   printf ("Down the server while files are open??\n");
   printf ("Y/N: ");
   ans = (char)getch();
   ans = toupper (ans);
   if (ans == 'Y')
      forceFlag = 1;
   if (ans == 'N')
      forceFlag = 0;
   else
   {
      printf ("INVALID CHOICE");
      exit (-1);
   }
   ccode = DownFileServer (forceFlag);
   switch (ccode)
   {
      case 0:
         printf ("SUCCESSFULLY downed the server...");
         break;

      case 198:
         printf ("WARNING!      WARNING!      WARNING!\n\n");
         printf("Console Operator rights required for this
                 utility\n");
         break;

      case 255:
         printf ("There are open files on the server!\n");
         break;

      default:
         printf ("Error %d in DownFileServer\n", ccode);
   }
}
```

# EnableFileServerLogin

Enables logins to the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWEnableFileServerLogin (page 39))

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int EnableFileServerLogin (void);
```

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

Server logins are disabled to restrict new accesses to a server during a crucial period of time (such as just before a server is shut down).

The requesting workstation must have console Operator rights. (See also: ENABLE LOGIN console command.)

## See Also

DisableFileServerLogin (page 402), GetFileServerLoginStatus (page 442)

## EnableFileServerLogin Example

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

main()
{
   int    ccode;
   char   serverName[48];
```

```
   WORD   connID;
   printf ("\n\n");
   ccode = EnableFileServerLogin ();
   if (ccode == 198)
      printf("Console Operator rights required for utility. \n");
   else
      if (ccode == 0)
      {
         connID = GetDefaultConnectionID ();
         GetFileServerName (connID, serverName);
         printf ("SUCCESSFUL\n\n");
         printf ("Enabled logins to %s\n", serverName);
      }
   else
   printf ("Completion Code  =   %d\n", ccode);
}
```

# EnableTransactionTracking

Enables transaction tracking on the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWEnableTTS (page 364))

**Local Servers:** blocking

**Remote Servers:** blocking

**NetWare Server:** 3.x

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int EnableTransactionTracking (void);
```

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

Transaction tracking might have been disabled explicitly calling DisableTransactionTracking or automatically by the server because the transaction volume is full. After freeing space on the transaction volume by deleting unneeded files, the operator can use this function to enable transaction tracking.

This function has no effect if TTS is not installed on the server.

The requesting workstation must have console Operator rights. (See also: ENABLE TTS console command.)

## See Also

DisableTransactionTracking (page 404)

## EnableTransactionTracking Example

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

```
main()
{
   int   ccode;
   printf ("\n\n");
   ccode = EnableTransactionTracking ();
   if (ccode == 0)
      printf ("SUCCESSFULLY enabled TRANSACTION TRACKING.\n");
   else
      printf ("ccode = %d\n", ccode);
}
```

# GetBinderyObjectDiskSpaceLeft

Returns an object's remaining disk space (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetObjectDiskSpaceLeft)

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int GetBinderyObjectDiskSpaceLeft (
   WORD   fileServerID,
   long   objectID,
   long   *systemElapsedTime,
   long   *unusedDiskBlocks,
   BYTE   *restrictionEnforced);
```

## Parameters

**`fileServerID`**

(IN) Specifies the connection ID of the server for which to get information.

**`objectID`**

(IN) Specifies the ID of the object for which to return information.

**`systemElapsedTime`**

(OUT) Specifies the time elapsed since the server was brought up.

**`unusedDiskBlocks`**

(OUT) Specifies the number of 4K blocks the object has left.

**`restrictionEnforced`**

(OUT) Indicates the limitations placed on disk resources; 0 indicates maximum disk space is enforced for the object by the server.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |

| Decimal | Hex | Constant |
|---|---|---|
| 198 | (0XC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

This function allows the requesting workstation to get remaining disk space for its own logged object.

The `objectID` parameter is a 4-byte identification number assigned to an object by the server. It uniquely identifies the object within the NetWare Directory.

The `systemElapsedTime` parameter indicates the time that has elapsed since the server was brought up. It is returned in units of approximately 1/18th of a second. This field can be used to determine the amount of time that has elapsed between consecutive calls. When this field reaches 0xFFFFFFFF, it wraps back to zero.

The `unusedDiskBlocks` parameter indicates the number of remaining 4K blocks the object can allocate. The `unusedDiskBlocks` available to a user are not related to how much free disk space is actually available.

The `restrictionsEnforced` parameter indicates the limitations placed on disk resources (0x00 = enforced, 0xFF = not enforced). For disk resource limitation to be active, it must be selected as an option during network installation. If it is not selected as an option, it is not enforced by the server.

Console Operator rights are required to get remaining disk space for other objects.

## See Also

GetDefaultFileServerID

## GetBinderyObjectDiskSpaceLeft Example

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

main()
{
   int    ccode, restrictionEnforced;
   int    days, hours, minutes, seconds;
   char   serverName[48];
   WORD   fileServerID, oldConnID;
   long   objectID, systemElapsedTime, unusedDiskBlocks;
   float   tmpTime;
   printf ("Enter server name:  ");
   scanf ("%s", serverName);
   GetfileServerID (serverName, &fileServerID);
   oldConnID = GetPreferredConnectionID ();
   SetPreferredConnectionID (fileServerID);

/*- Replace "JDOE" with the user name you want -*/
   GetBinderyObjectID ("JDOE", OT_USER, &objectID);
```

```
   ccode = GetBinderyObjectDiskSpaceLeft(fileServerID, objectID,
      &systemElapsedTime,  &unusedDiskBlocks, &restrictionEnforced);
   if (ccode == 0)
   {
      tmpTime =  ((((   ((float)(systemElapsedTime))/18)/60)/60)/24);
      days = (int)tmpTime;
      tmpTime = (tmpTime - days) * 24;
      hours = (int)tmpTime;
      tmpTime = (tmpTime - hours) * 60;
      minutes = (int)tmpTime;
      tmpTime = (tmpTime - minutes) * 60;
      seconds = (int)tmpTime;
      printf ("\n\n\n");
      printf ("Connection ID       %u\n", fileServerID);
      printf ("Object ID           %08lX\n", objectID);
      printf ("Elapsed Time        %d DAYS %d HOURS ", days, hours);
      printf ("%d MINUTES %d SECONDS\n", minutes, seconds);
      printf ("Unused Disk Blocks        %08lX\n", unusedDiskBlocks);
      printf ("Restriction Enforced??    %d\n", restrictionEnforced);
   }
   else
      printf ("Error %d in GetBinderyObjectDiskSpaceLeft\n", ccode);
   SetPreferredConnectionID (oldConnID);
}
```

# GetConnectionSemaphores

Returns information about a connection's open semaphores (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (NLM Development Concepts, Tools, and Functions) and call NWScanSemaphoresByConn in Single and Intra-File Management)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nlm\nit\nwenvrn1.h>

T_RC GetConnectionSemaphores (
   WORD              connectionNumber,
   int              *lastRecord,
   int              *lastTask,
   int               structSize,
   CONN_SEMAPHORE   *connectionSemaphore,
   void             *buffer,
   int               bufferSize);
```

## Parameters

**connectionNumber**

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

**lastRecord**

(IN/OUT) Points to the next major record (set to 0 for the first call).

**lastTask**

(IN/OUT) Points to the task number within the logical connection that has the semaphore open ( set to 0 for the first call).

**structSize**

(IN) Specifies the size of the CONN_SEMAPHORE structure.

**connectionSemaphore**

(OUT) Points to a structure that contains information about the connection's open semaphores.

**buffer**

(IN/OUT) Points to a buffer that is used during processing. If this function is called iteratively, `buffer` should be reused.

**bufferSize**

(IN) Specifies the size of `buffer`. The buffer should be at least ENVSERV_BUFFER1_SIZE bytes.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns information about the open semaphores for the connection specified by `connectionNumber`. The requesting connection must have console operator rights.

To call this function iteratively, an application must set `lastRecord` and `lastTask` to zero on the first call. The function changes `lastRecord` to point to the next major record, if any. If `lastRecord` and `lastTask` are returned as zero, there is no more information to receive.

The `structSize` parameter is the size of CONN_SEMAPHORE, which should be at least the size of CONN_SEMAPHORE_386 plus two bytes (for the WORD, `unionType`). The connection type (386 server) can be determined by calling CheckNetWareVersion.

GetConnectionSemaphores returns different information depending on whether the connection is to a NetWare 3.x (or above) server. The CONN_SEMAPHORE structure is defined in NWENVRN1.H as follows:

```
typedef struct CONN_SEMAPHORE
{
   WORD   unionType;
   union
   {
      CONN_SEMAPHORE_386 con386;
   }u;
}CONN_SEMAPHORE;
```

The `unionType` field indicates the type of information (386) that is returned. If the connection is to a NetWare 3.x (or above) server, `unionType` is ENVSERV_CONN_TYPE_386 and a CONN_SEMAPHORE_386 structure is returned. Both structures are defined in NWENVRN1.H as follows:

```
typedef struct CONN_SEMAPHORE_386
{
   WORD   openCount;
   WORD   semahoreValue;
   WORD   taskNumber;
   BYTE   nameLength;
   BYTE   semaphoreName[255];
} CONN_SEMAPHORE_386;
```

The first `structSize` bytes are copied into the structure.

The `openCount` field contains the number of logical connections that have this semaphore open.

The `semaphoreValue` field contains the current value of the semaphore. A negative value is usually interpreted as the number of processes waiting for the service represented by the semaphore.

The `taskNumber` field contains the task number within the logical connection that has the semaphore open.

The `semaphoreName` field contains a length-preceded string representing the semaphore name.

# GetConnectionsOpenFiles

Returns information about the files that a connection has open (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (NLM Development Concepts, Tools, and Functions) and call NWScanOpenFilesByConn2)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nlm\nit\nwenvrn1.h>

T_RC GetConnectionsOpenFiles (
   WORD               connectionNumber,
   int               *lastRecord,
   int               *lastTask,
   int                structSize,
   CONN_OPEN_FILES   *openFiles,
   void              *buffer,
   int                bufferSize);
```

## Parameters

**`connectionNumber`**

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

**`lastRecord`**

(IN/OUT) The function sets this parameter to point to the next major record (must be set to 0 for the first call).

**`lastTask`**

(IN/OUT) The function sets this parameter to point to the task number within the logical connection that has the semaphore open (must be set to 0 for the first call).

**`structSize`**

(IN) Specifies the size of the CONN_OPEN_FILES structure.

**`openFiles`**

(OUT) Points to a structure that contains information about files that the connection has open.

**`buffer`**

(IN/OUT) Points to a buffer that is used during processing. If this function is called iteratively, `buffer` should be reused.

**bufferSize**

(IN) Specifies the size of `buffer`. The buffer should be at least ENVSERV_BUFFER1_SIZE bytes.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns information about the open files of the connection specified by `connectionNumber`. The requesting connection must have console operator rights.

To call this function iteratively, an application must set `lastRecord` and `lastTask` to zero on the first call. The function changes `lastRecord` to point to the next major record, if any. If `lastRecord` and `lastTask` are returned as zero, there is no more information to receive.

The `structSize` parameter is the size of CONN_OPEN_FILES, which should be at least the size of CONN_OPEN_FILES_386 plus two bytes (for the WORD, `unionType`). The connection type (386 server) can be determined by calling CheckNetWareVersion.

GetConnectionsOpenFiles returns different information depending on whether the connection is to a NetWare 3.x (or above) server. The CONN_OPEN_FILES structure is defined in NWENVRN1.H as follows:

```
typedef struct CONN_OPEN_FILES
{
   WORD   unionType;
   union
   {
      CONN_OPEN_FILES_386 con386;
   }u;
} CONN_OPEN_FILES;
```

The `unionType` field indicates the type of information (386) that is returned. If the connection is to a NetWare 3.x (or above) server, `unionType` is ENVSERV_CONN_TYPE_386 and a CONN_OPEN_FILES_386 structure is returned. This structure is defined in NWENVRN1.H as follows:

```
typedef struct CONN_OPEN_FILES_386
{
   WORD   taskNumber;
   BYTE   lockType;
   BYTE   accessControl;
   BYTE   lockFlag;
   BYTE   volumeNumber;
   LONG   parentDirEntry;
   LONG   dirEntry;
   BYTE   forkCount;
   BYTE   nameSpace;
   BYTE   nameLength;
   BYTE   fileName[256];
}CONN_OPEN_FILES_386;
```

The first `structSize` bytes are copied to the structure.

The `taskNumber` field indicates the task number that has the file open.

The `lockType` field contains bit flags indicating the type of file lock as follows:

0 Locked
1 Open Shareable
2 Logged
3 Open Normal
6 TTS Holding Lock
7 Transaction Flag Set For This File

The `accessControl` field contains bit flags indicating the connection's access rights for the file as follows:

0 Open for read by this connection
1 Open for write by this connection
2 Deny read requests by other connections
3 Deny write requests by other connections
4 File detached
5 TTS holding detach
6 TTS holding open

The `lockFlag` field contains a flag indicating the type of lock on the file as follows:

| Decimal | Hex | Description |
|---|---|---|
| 0 | 0x00 | Not locked |
| 254 | 0xFE | Locked by a file lock |
| 255 | 0xFF | Locked by Begin Share File Set |

The `volumeNumber` field identifies the file's volume in a Volume Table on the server. The Volume Table contains information about each volume on the server.

The `dirEntry` field indicates the file path that is relative to this directory. This value is not a directory handle.

The `fileName` field contains an ASCII string representing the file's name.

CONN_OPEN_FILES_386 contains an additional four fields.

The `parentDirEntry` field contains the file path relative to the parent directory.

The `forkCount` field contains the index assigned (by Novell®) to the non-primary data stream associated with the file (for example, the resource fork of a Macintosh file).

The `nameSpace` field contains a number indicating the name space of the file. The name spaces currently available are:

| | |
|---|---|
| 0 | DOS |

| | |
|---|---|
| 1 | MACINTOSH |
| 2 | NFS |
| 3 | FTAM |
| 4 | OS2 |
| 5 | NT |

The `nameLength` field contains the length of `fileName`.

The `fileName` field contains the file name.

# GetConnectionsTaskInformation

Returns information about a connection's active tasks (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetTaskInformationByConn)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Synta

```
#include <nlm\nit\nwenvrn1.h>

T_RC GetConnectionsTaskInformation (
   WORD     connectionNumber,
   void   **connectionTaskInfo,
   void    *buffer,
   int      bufferSize);
```

## Parameters

**`connectionNumber`**

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

**`connectionTaskInfo`**

(OUT) Points to a structure containing task information for the connection.

**`buffer`**

(IN/OUT) Points to a buffer that receives CONN_SEMAPHORE_386.

**`bufferSize`**

(IN) Specifies the size of `buffer`. The buffer should be at least ENVSERV_BUFFER1_SIZE bytes.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns information about the active tasks for the connection identified by `connectionNumber`.

The `connectionTaskInfo` parameter receives a pointer to one or more structures. The structure(s) is CONN_TASK_INFO_386 depending on whether the connection is to a 3.x (or above)

server. If the structure is CONN_TASK_INFO_386, its `unionType` field contains ENVSERV_CONN_TYPE_386. These structures are defined in NWENVRN1.H as follows:

```
typedef struct CONN_TASK_INFO_386
{
   WORD   unionType;
   BYTE   lockStatus;
   union
   {
      struct
      {
         WORD   taskNumber;
         LONG   beginAddress;
         LONG   endAddress;
         WORD   volumeNumber;
         LONG   parentID;
         LONG   directoryEntry;
         BYTE   forkCount;
         BYTE   nameSpace;
         BYTE   nameLength;
         BYTE   name;
      }LockStatus1;
      struct
      {
         WORD   taskNumber;
         WORD   volumeNumber;
         LONG   parentID;
         LONG   directoryEntry;
         BYTE   forkCount;
         BYTE   nameSpace;
         BYTE   nameLength;
         BYTE   name;
      }LockStatus2;
      struct
      {
         WORD   taskNumber;
         BYTE   nameLength;
         BYTE   name;
      }LockStatus3Or4;
   }waitRecord;
}CONN_TASK_INFO_386;
```

The `lockStatus` field receives one of the following values:

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

This field indicates which of the following structures ( LockStatus1, LockStatus2, or LockStatus3Or4) is filled out. If `lockStatus` is 0, none of these structures is filled out.

The `taskNumber`, `nameLength`, and `name` fields are returned for all lock status types (except 0).

The `taskNumber` field contains the task number.

The `nameLength` field contains length of `name`.

The `name` field contains the name of the file that is locked.

LockStatus1 and LockStatus2 also contain `volumeNumber` and `directoryEntry` fields.

The `volumeNumber` field contains the volume number of the volume containing the file.

The `directoryEntry` field contains the directory entry of the file.

For the CONN_TASK_INFO_386 structure, the LockStatus1 and LockStatus2 structures also contain `parentID`, `forkCount`, and `nameSpace` fields.

The `parentID` field contains the directory entry for the parent directory.

The `forkCount` field contains the index assigned (by Novell) to the non-primary data stream associated with the file (for example, the resource fork of a Macintosh file).

The `nameSpace` field contains a number indicating the name space of the file. The five name spaces that are currently available are:

| | |
|---|---|
| 0 | DOS |
| 1 | MACINTOSH |
| 2 | NFS |
| 3 | FTAM |
| 4 | OS2 |
| 5 | NT |

The LockStatus1 structure also contains `beginAddress` and `endAddress` fields, which contain the starting address and the ending address of the locked region in the file, respectively.

Directly following the waitRecord is a byte indicating the number of active tasks for the connection. For each active task a structure follows. This structure is CONN_TASK_PAIRS_386 (for a connection to a NetWare 3.x or above server). This structure is defined in NWENVRN1.H as follows:

```
typedef struct CONN_TASK_PAIRS_386
{
   WORD   task;
   BYTE   taskStatus;
}CONN_TASK_PAIRS_386;
```

The `task` field contains the task number.

The `taskStatus` field contains the state of `task`:

| Decimal | Hex | State and Description |
|---|---|---|
| 1 | 0x01 | TState_TTSEXPLICIT: Indicates a TTS explicit transaction is in progress. |
| 2 | 0x02 | TState_TTSIMPLICIT: Indicates a TTS implicit transaction is in progress. |
| 4 | 0x04 | TState_FileSetLock: Indicates a Shared file set lock is in progress. |

## GetConnectionsUsageStats (obsolete 4/99)

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

# GetConnectionsUsingFile

Returns all logical connections using a file (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (NLM Development Concepts, Tools, and Functions) and call NWScanConnectionsUsingFile in Multiple and Inter-File Management)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nlm\nit\nwenvrn1.h>

T_RC GetConnectionsUsingFile (
   int     requestSize,
   void   *request,
   void   *buffer,
   int     bufferSize);
```

## Parameters

**`requestSize`**

(IN) Specifies the size of the request buffer.

**`request`**

(IN) Points to a buffer containing the request structure.

**`buffer`**

(IN/OUT) Points to a buffer which receives the response structure.

**`bufferSize`**

(IN) Specifies the size of the buffer that receives the response structure. The buffer should be at least ENVSERV_BUFFER1_SIZE bytes.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function uses a different request structure depending on whether the request is made to a NetWare 3.x (or above) server. You must determine the type of server that the request is sent to and put the correct request structure into the `request` buffer. If the wrong structure is submitted, this function returns EFAILURE.

The buffer receives the response structure.

The request buffer contains a CONN_USING_FILE_REQUEST structure, defined in NWENVRN1.H as follows:

```
typedef   struct CONN_USING_FILE_REQUEST
{
   WORD   unionType;
   WORD   reserved1;
   BYTE   reserved2;
   union
   {
      CONN_USING_FILE_REQ_386 req386;
   };
}CONN_USING_FILE_REQUEST;
```

The unionType field contains ENVSERV_CONN_TYPE_386 (for a NetWare 3.x or above server).

The union field contains the request structure for the server type that you want information for.

*NetWare 3.x (or above) Server:* The request structure you send to NetWare 3.x (or above) servers is CONN_USING_FILE_REQ_386, defined in NWENVRN1.H as follows:

```
typedef   struct CONN_USING_FILE_REQ_386
{
   BYTE   forkType;
   BYTE   volume;
   LONG   directoryID;
   WORD   nextRecord;
}CONN_USING_FILE_REQ_386;
```

The forkType field contains the index assigned (by Novell) to the non-primary data stream associated with the file (for example, the resource fork of a Macintosh file).

The volume field contains the volume that the file is on.

The directoryID field contains the directory handle of the directory that contains the file.

The nextRecord field identifies the first record to retrieve data for. This field must be set to 0 for the first call to this function. On subsequent calls, it should be given the value returned in the nextRequestRecord field of the reply structure.

The buffer receives a CONN_USING_FILE_REPLY structure, defined in NWENVRN1.H as follows:

```
typedef struct CONN_USING_FILE_REPLY
{
   WORD   unionType;
   union
   {
      struct CONN_USING_FILE_REPLY_386 rep386;
   };
}CONN_USING_FILE_REPLY;
```

The unionType field contains ENVSERV_CONN_TYPE_386 (for a NetWare 3.x or above server).

The union field contains the reply structure for the server type that you want information for.

*NetWare 3.x (or above) Server:* The buffer receives a CONN_USING_FILE_REPLY_386 structure followed by CONN_USING_FILE_RECORD_386 structures. The CONN_USING_FILE_REPLY_386 structure is defined in NWENVRN1.H as follows:

```
typedef struct CONN_USING_FILE_REPLY_386
{
   WORD   nextRequestRecord;
   WORD   useCount;
   WORD   openCount;
   WORD   openForReadCount;
   WORD   openForWriteCount;
   WORD   denyReadCount;
   WORD   denyWriteCount;
   BYTE   locked;
   BYTE   forkCount;
   WORD   numberOfRecords;   /* connection records follow */
}CONN_USING_FILE_REPLY_386;
```

The nextRequestRecord field contains a value to be passed in the nextRecord field of CONN_USING_FILE_REQ_386 for the next call to this function. This field contains 0 when the last record is retrieved and no more calls to this function need to be made.

In addition, this structure has a forkCount field, which contains the index assigned (by Novell) to the non-primary data stream associated with the file (for example, the resource fork of a Macintosh file).

The reply structure is followed by a number of structures, one for each connection using the file.

*NetWare 3.x (or above) Server:* The CONN_USING_FILE_REPLY_386 structure is followed by a number of CONN_USING_FILE_RECORD_386 structures, one for each connection using the file. This structure is defined in NWENVRN1.H as follows:

```
typedef   struct CONN_USING_FILE_RECORD_386
{
   WORD   connectionNumber;
   WORD   taskNumber;
   BYTE   lockType;
   BYTE   accessFlags;
   BYTE   lockStatus;
}CONN_USING_FILE_RECORD_386;
```

The connectionNumber field contains the connection number using the file.

The taskNumber field contains the connection's task number that is using the file.

The lockType field contains bit flags indicating the file's lock information, as shown below:

0 Locked
1 Open Shareable
2 Logged
3 Open Normal
6 TTS Holding Lock
7 Transaction Flag Set For This File

The `accessFlags` field contains bit flags indicating the connection/task's access rights for the file, as shown below:

0 Open for read by this connection
1 Open for write by this connection
2 Deny read requests by other connections
3 Deny write requests by other connections
4 File detached
5 TTS holding detach
6 TTS holding open

The `lockStatus` field contains a flag indicating the type of lock, if any, on the file, as follows:

| Decimal | Hex | Description |
|---|---|---|
| 0 | 0x00 | Not locked |
| 254 | 0xFE | Locked by a file lock |
| 255 | 0xFF | Locked by Begin Share File Set |

# GetDiskCacheStats (obsolete 4/99)

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

# GetDiskChannelStats (obsolete 4/99)

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

# GetDiskUtilization

Returns the disk usage of an object on a volume (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (NLM Development Concepts, Tools, and Functions) and call NWGetDiskUtilization in Volume Management)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int GetDiskUtilization (
   long    objectID,
   BYTE    volumeNumber,
   LONG   *usedDirectories,
   LONG   *usedFiles,
   LONG   *usedBlocks);
```

## Parameters

**objectID**

(IN) Specifies the unique ID of the object.

**volumeNumber**

(IN) Specifies the volume for which statistics are requested (0-based).

**usedDirectories**

(OUT) Receives the number of directories owned by the object.

**usedFiles**

(OUT) Receives the number of files created by the object.

**usedBlocks**

(OUT) Receives the number of disk blocks used by the object.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 152 | (0x98) | ERR_VOLUME_DOES_NOT_EXIST |

| Decimal | Hex | Constant |
|---|---|---|
| 242 | (0xF2) | ERR_NO_OBJECT_READ_PRIVILEGE |

## Remarks

This function returns the disk usage of an object by passing the object ID.

The `volumeNumber` parameter identifies the volume in a volume table on the server. The volume table contains information about each volume on the server.

The `objectID` parameter is 4-byte identification number assigned to an object by the server. It uniquely identifies the object within the NetWare Directory.

To determine the number of bytes an object is using on a volume, use the following equation:

```
bytes used =  usedBlocks
 * sectors/block * bytes/sector
```

Current network implementations allocate eight 512-byte sectors per block (4 KB per block).

This function requires Read privileges at the object level.

## See Also

GetVolumeInformation (NetWare SDK)

## GetDiskUtilization Example

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

main()
{
   int    ccode, volumeNumber;
   LONG   objectID, usedDirectories, usedFiles, usedBlocks;
   WORD   objType;
   char   objName[48], volName[16];
   printf ("\n\n");
   printf ("Enter object's name: ");
   scanf ("%s", objName);
   printf ("\nEnter object's type: ");
   scanf ("%d", &objType);
   ccode = GetBinderyObjectID (objName, objType, &objectID);
   printf ("\nEnter volume name:   ");
   scanf ("%s", volName);
   ccode = GetVolumeNumber (volName, &volumeNumber);
   ccode = GetDiskUtilization (objectID, *((BYTE *)volumeNumber),
      &usedDirectories, &usedFiles, &usedBlocks);
   if (ccode)
   {
      if (ccode == 152)
      printf ("\n\nVOLUME DOES NOT EXIST\n");
```

```
      else if (ccode == 242)
      printf ("\n\nNO OBJECT READ PRIVILEGE\n");
      else
      printf ("\n\nccode = %d\n", ccode);
   }
   else
   {
      printf ("\n\n\nSUCCESSFULLY COMPLETED\n\n\n");
      printf ("Object ID...                %08lX\n", objectID);
      printf ("Volume Number...            %d\n", volumeNumber);
      printf ("Directories Used...         %u\n", usedDirectories);
      printf ("Files Used...               %u\n", usedFiles);
      printf ("Blocks Used...              %u\n", usedBlocks);
   }
}
```

## GetDriveMappingTable (obsolete 4/99)

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

# 11.2 GetF*-TTS* Functions

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

# GetFileServerDateAndTime

Returns the date and time of the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetFileServerDateAndTime (page 59))

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

void GetFileServerDateAndTime (
   BYTE   *dateAndTime);
```

## Parameters

**`dateAndTime`**

(OUT) Receives the server's date and time (7 bytes).

## Remarks

The `dateAndTime` parameter returns information in the following format:

| Byte | Contents | Values | Explanation |
|---|---|---|---|
| 0 | Year | 80 to 179 | 80-99 correspond to the years 1980-1999. 100-179 correspond to the years 2000-2079. |
| 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) | A value of 0 = Sunday, 1 = Monday, 2 = Tuesday, etc. |

The date and time are not synchronized across the network (between servers) unless time synchronization is active.

## See Also

GetClockStatus (NetWare SDK), SetFileServerDateAndTime (page 476)

## GetFileServerDateAndTime Example

```
#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <nwtypes.h>
#include <nlm\nit\nwenvrn.h>

main()
{
   int    rc;
   char   d[7];
   GetFileServerDateAndTime(d);
   printf("year = %d\n",d[0]);
   printf("month = %d\n",d[1]);
   printf("day = %d\n",d[2]);
   printf("hour = %d\n",d[3]);
   printf("minute = %d\n",d[4]);
   printf("second = %d\n",d[5]);
   printf("weekday = %d\n",d[6]);
}
```

# GetFileServerDescriptionStrings

Returns the name of the company that distributed this copy of NetWare (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetFileServerDescription (page 61))

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int GetFileServerDescriptionStrings (
   char   *companyName,
   char   *revision,
   char   *revisionDate,
   char   *copyrightNotice);
```

## Parameters

**companyName**

(OUT) Receives a string containing the name of the company (80 characters, including the NULL terminator).

**revision**

(OUT) Receives a string containing the NetWare version and revision (80 characters, including the NULL terminator).

**revisionDate**

(OUT) Receives a string containing the revision date in the form MM/DD/YY (up to 24 characters, including the NULL terminator).

**copyrightNotice**

(OUT) Receives a string containing the copyright notice (maximum 80 characters, including the NULL terminator).

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |

## Remarks

The `companyName`, `revision`, `revisionDate`, and `copyrightNotice` parameters return information about this copy of NetWare.

## See Also

GetServerInformation (page 468)

## GetFileServerDescriptionStrings Example

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

main()
{
   int    ccode;
   char   companyName[80], revision[80];
   char   revisionDate[24], copyrightNotice[80];
   ccode = GetFileServerDescriptionStrings (companyName, revision,
               revisionDate, copyrightNotice);
   if (ccode == 0)
   {
      printf ("\n\n\n");
      printf ("%s\n", companyName);
      printf ("%s\n", revision);
      printf ("%s\n", revisionDate);
      printf ("%s\n", copyrightNotice);
   }
}
```

# GetFileServerLANIOStats (obsolete 4/99)

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

# GetFileServerLoginStatus

Returns the server's login status (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetFileServerLoginStatus (page 71))

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int GetFileServerLoginStatus (
   int   *loginEnabledFlag);
```

## Parameters

**loginEnabledFlag**

(OUT) Receives a flag indicating the status of the server's login state (0 = Login is disabled).

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

The login status indicates whether logins are enabled or disabled.

Server logins can be disabled with the DisableFileServerLogin function so that new accesses to a server are restricted during a crucial period of time (before taking a server down). The EnableFileServerLogin function re-enables logins.

The requesting workstation must have console Operator rights.

## See Also

DisableFileServerLogin (page 402), EnableFileServerLogin (page 408)

## GetFileServerLoginStatus Example

```
#include <stdio.h>
#include <stdlib.h>
#include <nlm\nit\nwenvrn.h>

main()
{
   int   ccode, loginEnabledFlag;
   ccode = GetFileServerLoginStatus (&loginEnabledFlag);
   if (ccode)
   {
      printf ("ccode = %d\n", ccode);
      exit (-1);
   }
   else
      if (loginEnabledFlag)
      printf ("Login is enabled.\n");
      else
      printf ("Login is disabled.\n");
   }
}
```

# GetFileServerMiscInformation (obsolete 4/99)

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

# GetFileServerName

Returns the name of a server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetFileServerVersionInfo (page 74) or NWCCGetConnInfo)

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

void GetFileServerName (
   WORD    fileServerID,
   char   *fileServerName);
```

## Parameters

**`fileServerID`**

(IN) Contains the connection ID for which `fileServerName` is returned; if this number is 0, the name of the server is returned.

**`fileServerName`**

(OUT) Receives name of the server (maximum 48 characters).

## Return Values

This function returns no value. If an error occurs, `NetWareErrno` is set to:

| Decimal | Hex | Constant |
|---|---|---|
| 115 | (0x73) | ERR_BAD_CONNECTION_ID |

## Remarks

This function returns the name of a server by passing the `fileServerID`. If `fileServerID` is invalid, `fileServerName` is NULL.

Passing the `fileServerID` (server number) of any logged in remote server returns the name of that server. If `fileServerID` is 0, the name of the local server is returned.

## See Also

GetFileServerID, GetServerInformation (page 468)

## GetFileServerName Example

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

main()
{
   WORD   fileServerID;
   char   fileServerName[48];
   printf ("\n\n");
   printf ("Enter Connection ID: ");
   scanf ("%u", &fileServerID);
   GetFileServerName (fileServerID, fileServerName);
   printf ("Connection ID is: %u\n", fileServerID);
   printf ("Server Name is: %s\n", fileServerName);
}
```

# GetFileSystemStats (obsolete 4/99)

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

# GetLANDriverConfigInfo (obsolete 4/99)

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

# GetLogicalRecordInformation

Returns information about a logical record (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (NLM Development Concepts, Tools, and Functions) and call NWScanLogicalLocksByName in Single and Intra-File Management)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nlm\nit\nwenvrn1.h>

T_RC GetLogicalRecordInformation (
   int     requestSize,
   void   *request,
   void   *buffer,
   int     bufferSize);
```

## Parameters

**requestSize**

(IN) Specifies the size of the request buffer.

**request**

(IN) Points to a buffer containing the request structure.

**buffer**

(IN/OUT) Points to a buffer which receives the response structure.

**bufferSize**

(IN) Specifies the size of the buffer that receives the response structure. This buffer must be at least ENVSERV_BUFFER1_SIZE bytes.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

The `request` buffer contains a CONN_USING_FILE_REQUEST structure, defined in NWENVRN1.H as follows:

```
typedef struct LOGICAL_RECORD_REQUEST
{
```

```
   WORD   reserved1;
   BYTE   reserved2;
   WORD   nextRecord;
   BYTE   nameLength;
   BYTE   name[255];
}LOGICAL_RECORD_REQUEST;
```

The `nextRecord` field identifies the first record to be processed by this function. This field must be set to 0 for the first call to this function. On subsequent calls, it must be set to the value returned in the `nextRequestRecord` field of the reply structure.

The `nameLength` field contains the length of `name`.

The `name` field contains the name of the logical record that you want information about.

The `buffer` parameter receives the response structure, LOGICAL_RECORD_INFO, which is defined in NWENVRN1.H as follows:

```
typedef struct LOGICAL_RECORD_INFO
{
   WORD   unionType;
   union
   {
      LOGICAL_RECORD_INFO_386 lr386;
   }u;
}LOGICAL_RECORD_INFO;
```

The `unionType` field indicates the type of information (386) that is returned. If the connection is to a NetWare 3.x (or above) server, `unionType` is ENVSERV_CONN_TYPE_386 and a LOGICAL_RECORD_INFO_386 structure is returned. Both structures are defined in NWENVRN1.H as follows:

```
typedef struct LOGICAL_RECORD_INFO_386
{
   WORD   useCount;
   WORD   shareableLockCount;
   BYTE   locked;
   WORD   nextRequestRecord;
   WORD   numberOfRecords;
}LOGICAL_RECORD_INFO_386;
```

The `useCount` field contains the number of logical connections that have the logical record logged.

The `sharableLockCount` field contains the number of logical connections that have a shareable lock on the logical record.

The `locked` field indicates whether the logical record is locked exclusively (0 = not locked exclusively).

The `nextRequestRecord` field identifies the next record to be processed by this function. This value is passed in the `nextRecord` field of the request function on the next call to this function. When 0 is returned in this field, there are no more records to be processed.

The `numberOfRecords` field contains the number of structures that follow in the buffer (LOGICAL_RECORD_386 structures). This structure is defined in NWENVRN1.H as follows:

```
typedef struct LOGICAL_RECORD_386
{
   WORD   connectionNumber;
   WORD   taskNumber;
   BYTE   lockStatus;
}LOGICAL_RECORD_386;
```

The `connectionNumber` field contains the connection number of the connection using the record.

The `taskNumber` field contains the number of the task within the connection that is using the record.

The `lockStatus` field contains the bit flags that indicate the record's lock status, as follows:

0 Locked
1 Open Shareable
2 Logged
3 Open Normal
6 TTS Holding Lock
7 Transaction Flag Set For This File

# GetLogicalRecordsByConnection

Returns the logical records that a connection has logged with a server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (NLM Development Concepts, Tools, and Functions) and call NWScanLogicalLocksByConn (in Single and Intra-File Management)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nlm\nit\nwenvrn1.h>

T_RC GetLogicalRecordsByConnection (
   WORD    connectionNumber,
   WORD    nextRecord,
   void   *buffer,
   int     bufferSize);
```

## Parameters

**`connectionNumber`**

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

**`nextRecord`**

(IN) Specifies the first record to process. This parameter must be set to 0 for the first call to this function. On subsequent calls, it must be set to the value returned in the `nextRequest` field of the reply structure.

**`buffer`**

(IN/OUT) Points to a buffer which receives the reply structure.

**`bufferSize`**

(IN) Specifies the size of the buffer that receives the reply structure. This buffer must be at least ENVSERV_BUFFER1_SIZE bytes.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns information about logical records that a given connection has logged with a server.

This function returns a CONN_LOGICAL_RECORD structure in `buffer`. This structure is defined in NWENVRN1.H as follows:

```
typedef struct CONN_LOGICAL_RECORD
{
   WORD   unionType;
   union
   {
      CONN_LOGICAL_RECORD_386 lr386;
   }u;
}CONN_LOGICAL_RECORD;
```

The `unionType` field indicates the type of information (386) that is returned. If the connection is to a NetWare 3.x (or above) server, `unionType` is ENVSERV_CONN_TYPE_386 and a CONN_LOGICAL_RECORD_386 structure is returned. This structure is defined in NWENVRN1.H as follows:

```
typedef struct CONN_LOGICAL_RECORD_386
{
   WORD   nextRequest;
   WORD   numberOfRecords;
}CONN_LOGICAL_RECORD_386;
```

The `nextRequestRecord` field identifies the next record to be processed by this function. This value is passed in the `nextRecord` parameter for the next call. When this field is 0, there are no more records to be processed.

The `numberOfRecords` field contains the number of structures that follow in the buffer (LOGICAL_RECORD_BLOCK_386). This structure is defined in NWENVRN1.H as follows:

```
typedef struct CONN_LOGICAL_RECORD_BLOCK_386
{
   WORD   taskNumber;
   BYTE   lockStatus;
   BYTE   lockNameLength;
   BYTE   lockName;         /* 1st byte - more follow */
}CONN_LOGICAL_RECORD_BLOCK_386;
```

The `taskNumber` field contains the number of the task within the connection that has the record logged.

The `lockStatus` field contains the bit flags indicating the record's lock status as follows:

0 Locked
1 Open Shareable
2 Logged
3 Open Normal
6 TTS Holding Lock
7 Transaction Flag Set For This File

The `lockNameLength` field contains the length of `lockName`.

The `lockName` field contains the name of the lock.

# GetPathFromDirectoryEntry

Accesses a file path listed in a server's Directory Entry Table (DET) (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*).)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nlm\nit\nwenvrn1.h>

int GetPathFromDirectoryEntry (
   BYTE    volumeNumber,
   WORD    directoryEntry,
   BYTE   *pathLength,
   char   *path);
```

## Parameters

**`volumeNumber`**

(IN) Specifies the volume number of the volume containing the directory from which the file path is accessed.

**`directoryEntry`**

(IN) Specifies the offset into the Directory Entry Table of the file path to be accessed.

**`pathLength`**

(OUT) Receives the length of `path`.

**`path`**

(OUT) Receives the file path of the directory (ASCIIZ string).

## Return Values

ESUCCESS or NetWare errors.

## Remarks

An application must call GetConnectionsOpenFiles to get the offset into the server's DET of the file path to be accessed before calling GetPathFromDirectoryEntry.

## See Also

GetConnectionsOpenFiles (page 418)

# GetPhysicalDiskStats (obsolete 4/99)

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

# GetPhysicalRecordLocksByFile

Returns physical records that are locked in a file (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (NLM Development Concepts, Tools, and Functions) and call NWScanPhysicalLocksByFile (Single and Intra-File Management

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nlm\nit\nwenvrn1.h>

T_RC GetPhysicalRecordLocksByFile (
   int     requestSize,
   void   *request,
   void   *buffer,
   int    *bufferSize);
```

## Parameters

**`requestSize`**

(IN) Specifies the size of the request buffer.

**`request`**

(IN) Points to a buffer containing the request structure.

**`buffer`**

(IN/OUT) Points to a buffer which receives the response structure.

**`bufferSize`**

(IN) Specifies the size of the buffer that receives the response structure. This buffer must be at least ENVSERV_BUFFER1_SIZE bytes.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function uses a different request structure depending on whether the request is made to a NetWare 3.x (or above) server. You must determine the type of server that the request is sent to and put the correct request structure into the `request` buffer. If the wrong structure is submitted, this function returns EFAILURE.

The buffer parameter receives the response structure.

The request buffer contains a FILE_PHYSICAL_RECORD_REQUEST structure, defined in NWENVRN1.H as follows:

```
typedef struct FILE_PHYSICAL_RECORD_REQUEST
{
   WORD   unionType;
   WORD   reserved1;
   BYTE   reserved2;
   union
   {
      FILE_PHYSICAL_REQUEST_386 pr386;
   }u;
}   FILE_PHYSICAL_RECORD_REQUEST;
```

The unionType field contains ENVSERV_CONN_TYPE_386 (for a NetWare 3.x or above server).

The union field contains the request structure for the server type that you want information for.

*NetWare 3.x (or above) Server:* The request for a NetWare 3.x (or above) server contains a FILE_PHYSICAL_REQUEST_386 structure, defined in NWENVRN1.H as follows:

```
typedef struct FILE_PHYSICAL_REQUEST_386
{
   BYTE   forkType;
   BYTE   volume;
   LONG   directoryID;
   WORD   next;
}FILE_PHYSICAL_REQUEST_386;
```

The forkType field contains the index assigned (by Novell) to the non-primary data stream associated with the file (for example, the resource fork of a Macintosh file).

The volume field contains the volume number of the file to return information for.

The directoryID field contains the directory ID number of the file.

The next field identifies the next record to be processed by this function. It must be set to 0 on the first call to this function. On subsequent calls, it must be set to the value returned in the nextRequest field of the reply structure.

The buffer receives a FILE_PHYSICAL_RECORD_LOCK structure, defined in NWENVRN1.H as follows:

```
typedef struct FILE_PHYSICAL_RECORD_LOCK
{
   WORD unionType;
   union
   {
      FILE_PHYSICAL_RECORD_LOCK_386 pr386;
   }u;
}FILE_PHYSICAL_RECORD_LOCK;
```

The unionType field contains ENVSERV_CONN_TYPE_386 (for a NetWare 3.x or above server).

The union field contains the reply structure for the server type that you want information for. The reply structure is FILE_PHYSICAL_RECORD_LOCK_386 for a NetWare 3.x (or above) server. This structure is defined in NWENVRN1.H as follows:

```
typedef struct FILE_PHYSICAL_RECORD_LOCK_386
{
   WORD   nextRequest;
   WORD   numberOfLocks;
}FILE_PHYSICAL_RECORD_LOCK_386;
```

The nextRequest field identifies the next record to be processed by this function. On subsequent calls to this function, this value is passed in the lastRecord field of the request structure. When 0 is returned in this field, all records have been processed.

The numberOfLocks field contains the number of lock information structures which follow in the buffer. If information for a NetWare 3.x (or above) server is requested, FILE_PHYSICAL_RECORD_386 structures follow. This structure is defined in NWENVRN1.H as follows:

```
typedef struct FILE_PHYSICAL_RECORD_386
{
   WORD   loggedCount;
   WORD   shareLockCount;
   LONG   recordStart;
   LONG   recordEnd;
   WORD   connectionNumber;
   WORD   taskNumber;
   BYTE   lockType;
}FILE_PHYSICAL_RECORD_386;
```

The loggedCount field contains the number of physical record locks.

The shareLockCount field contains the number of tasks that have the record locked shareable.

The recordStart field contains the starting byte offset of the physical record lock within the file.

The recordEnd field contains the ending byte offset of the physical record lock within the file.

The connectionNumber field contains the number of the connection that has the record locked exclusively.

The taskNumber field contains the task number of the task within the connection that has the record locked exclusively.

The lockType field contains a flag indicating the type of lock, if any, on the file, as shown below:

| Decimal | Hex | Description |
|---|---|---|
| 0 | 0x00 | Not locked |
| 254 | 0xFE | Locked by a file lock |
| 255 | 0xFF | Locked by Begin Share File Set |

The requesting connection must have console operator rights.

## See Also

GetPhysRecLockByConnectAndFile (page 462)

# GetPhysRecLockByConnectAndFile

Returns a logical connection's physical record locks within a file (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (NLM Development Concepts, Tools, and Functions) and call NWScanPhysicalLocksByConnFile (Single and Intra-File Management)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nlm\nit\nwenvrn1.h>

T_RC GetPhysRecLockByConnectAndFile (
   int    requestSize,
   void   *request,
   void   *buffer,
   int    *bufferSize);
```

## Parameters

**`requestSize`**

(IN) Specifies the size of the request buffer.

**`request`**

(IN) Points to a buffer containing the request structure.

**`buffer`**

(IN/OUT) Points to a buffer which receives the response structure.

**`bufferSize`**

(IN) Specifies the size of the buffer that receives the response structure. This buffer must be at least ENVSERV_BUFFER1_SIZE bytes.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function uses a different request structure depending on whether the request is made to a NetWare 3.x (or above) server. You must determine the type of server that the request is sent to and put the correct request structure into the `request` buffer. If the wrong structure is submitted, this function returns EFAILURE.

The buffer parameter receives the response structure.

The request buffer contains a CONN_LOCK_REQUEST structure, defined in NWENVRN1.H as follows:

```
typedef struct CONN_LOCK_REQUEST
{
   WORD   unionType;
   WORD   reserved1;
   BYTE   reserved2;
   union
   {
      CONN_LOCK_REQUEST_386 lr386;
   }u;
}CONN_LOCK_REQUEST;
```

The unionType field contains ENVSERV_CONN_TYPE_386 (for a NetWare 3.x or above server).

The union field contains the request structure for the server type that you want information for.

*NetWare 3.x (or above) Server:* The request for a NetWare 3.x (or above) server contains a CONN_LOCK_REQUEST_386 structure, defined in NWENVRN1.H as follows:

```
typedef struct CONN_LOCK_REQUEST_386
{
   WORD   connectionNumber;
   BYTE   forkType;
   BYTE   volume;
   LONG   directoryID;
   WORD   next;
}CONN_LOCK_REQUEST_386;
```

The connectionNumber field contains the connection number to obtain information for.

The forkType field contains the index assigned (by Novell) to the nonprimary data stream associated with the file (for example, the resource fork of a Macintosh file).

The volume field contains the number of the volume that contains the file.

The directoryID field contains the ID number of the directory that contains the file.

The next field identifies the next record to be processed by this function. On the first call, this field must be set to 0. On subsequent calls, it must be set to the value returned in the nextRecord field of the reply structure.

The buffer receives a CONN_RECORD_LOCKS structure, defined in NWENVRN1.H as follows:

```
typedef struct CONN_RECORD_LOCKS
{
   WORD   unionType;
   union
   {
      CONN_RECORD_LOCKS_386 rl386;
   }u;
}CONN_RECORD_LOCKS;
```

The `unionType` field contains ENVSERV_CONN_TYPE_386 (for a NetWare 3.x or above server).

The `union` field contains the reply structure for the server type that you want information for. The reply structure is CONN_RECORD_LOCKS_386 for a NetWare 3.x (or above) server. This structure is defined in NWENVRN1.H as follows:

```
typedef struct CONN_RECORD_LOCKS_386
{
   WORD   nextRecord;
   WORD   numberOfLocks;       /* record locks follow */
}CONN_RECORD_LOCKS_386;
```

The `nextRecord` field identifies the next record to be processed by this function. This value is passed to the next call in `next` (for NetWare 3.x) field of the request structure. When 0 is returned in this field, all records have been processed.

The `numberOfLocks` field contains the number of lock information structures which follow in the buffer. If information for a NetWare 3.x (or above) server is requested, CONN_LOCK_RECORD_386 structures follow. These structures are defined in NWENVRN1.H as follows:

```
typedef struct CONN_LOCK_RECORD_386
{
   WORD   taskNumber;
   BYTE   lockFlag;
   LONG   recordStart;
   LONG   recordEnd;
}CONN_LOCK_RECORD_386;
```

The `taskNumber` field contains the task number of the task within the connection that has the record locked exclusively.

The `lockFlag` field contains the bit flags indicating the record's lock status as follows:

0 Locked
1 Open Shareable
2 Logged
3 Open Normal
6 TTS Holding Lock
7 Transaction Flag Set For This File

The `recordStart` field contains the byte offset of the physical record lock within the file.

The `recordEnd` field contains the byte offset of the physical record lock within the file.

The requesting connection must have console operator rights.

## See Also

GetPhysicalRecordLocksByFile (page 458)

# GetSemaphoreInformation

Returns information about a semaphore (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions (NLM Development Concepts, Tools, and Functions) and call NWExamineSemaphore in Single and Intra-File Management)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nlm\nit\nwenvrn1.h>

T_RC GetSemaphoreInformation (
   int    requestSize,
   void   *request,
   void   *buffer,
   int    *bufferSize);
```

## Parameters

**`requestSize`**

(IN) Specifies the size of the request buffer.

**`request`**

(IN) Points to a buffer containing the request structure.

**`buffer`**

(IN/OUT) Points to a buffer which receives the response structure.

**`bufferSize`**

(IN) Specifies the size of the buffer that receives the response structure. This buffer must be at least ENVSERV_BUFFER1_SIZE bytes.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function uses a different request structure depending on whether the request is made to a NetWare 3.x (or above) server. You must determine the type of server that the request is sent to and put the correct request structure into the `request` buffer. If the wrong structure is submitted, this function returns EFAILURE.

The buffer parameter receives the response structure.

The request buffer contains a SEMAPHORE_INFO_REQUEST structure, defined in NWENVRN1.H as follows:

```
typedef struct SEMAPHORE_INFO_REQUEST
{
   WORD   reserved1;
   BYTE   reserved2;
   WORD   nextRecord;
   BYTE   nameLength;
   BYTE   name[255];
}SEMAPHORE_INFO_REQUEST;
```

The nextRecord field identifies the next record to be processed by this function. This field must be set to 0 on the first call to this function. On subsequent calls, it must be set to the value returned in the nextRequest field of the reply structure.

The nameLength field contains the length of name.

The name field contains the name of the semaphore to obtain information for.

The buffer receives a SEMAPHORE_INFO structure, defined in NWENVRN1.H as follows:

```
typedef struct SEMAPHORE_INFO
{
   WORD   unionType;
   union
   {
      SEMAPHORE_INFO_386 si386;
   }u;
}SEMAPHORE_INFO;
```

The unionType field contains ENVSERV_CONN_TYPE_386 (for a NetWare 3.x or above server).

The union field contains the reply structure for the server type that you want information for. The reply structure is SEMAPHORE_INFO_386 for a NetWare 3.x (or above) server. This structure is defined in NWENVRN1.H as follows:

```
typedef struct SEMAPHORE_INFO_386
{
   WORD   nextRequest;
   WORD   openCount;
   WORD   semaphoreValue;
   WORD   numberOfRecords;
}SEMAPHORE_INFO_386;
```

The nextRequest field identifies the next record to be processed by this function. This value is passed in the nextRecord field of the request structure on the next call to this function. When 0 is returned in this field, all records have been processed.

The openCount field contains the number of logical connections that have this semaphore open.

The semaphoreValue field contains the current value of the semaphore. A negative value is usually interpreted as the number of objects waiting for the service represented by the semaphore. A

positive value is usually interpreted as the number of free resources available in the resource pool governed by the semaphore.

The `numberOfRecords` field contains the number of semaphore information records that follow in the reply buffer. These records are SEMAPHORE_INFO_RECORD_386 structures for NetWare 3.x (or above) servers. This structure is defined in NWENVRN1.H as follows:

```
typedef struct SEMAPHORE_INFO_RECORD_386
{
   WORD   connectionNumber;
   WORD   taskNumber;
}SEMAPHORE_INFO_RECORD_386;
```

The `connectionNumber` field contains the connection number of the connection using the semaphore.

The `taskNumber` field contains the task number of the task within the connection that is using the semaphore.

# GetServerInformation

Returns information about the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetFileServerVersionInfo (page 74))

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int GetServerInformation (
   int               structSize,
   FILE_SERV_INFO   *serverInfo);
```

## Parameters

**`structSize`**

(IN) Specifies the number of bytes to return in `serverInfo`.

**`serverInfo`**

(OUT) Receives the statistics of the server (maximum 128 bytes).

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |

## Remarks

This function allows the user to retrieve all or part of the information about the server where the current connection is logged in. The function returns up to `structSize` bytes in `serverInfo`.

The FILE_SERV_INFO structure that `serverInfo` points to has the following definition:

```
{
   char    serverName[48];
   BYTE    netwareVersion;
   BYTE    netwareSubVersion;
   WORD    maxConnectionsSupported;
   WORD    connectionsInUse;
```

```
   WORD   maxVolumesSupported;
   BYTE   revisionLevel;
   BYTE   SFTLevel;
   BYTE   TTSLevel;
   WORD   peakConnectionsUsed;
   BYTE   accountingVersion;
   BYTE   VAPversion;
   BYTE   queingVersion;
   BYTE   printServerVersion;
   BYTE   virtualConsoleVersion;
   BYTE   securityRestrictionLevel;
   BYTE   internetBridgeSupport;
   BYTE   reserved[60];
   BYTE   CLibMajorVersion;
   BYTE   CLibMinorVersion;
   BYTE   CLibRevision;
}   FILE_SERV_INFO;
```

`netwareVersion` and `netwareSubVersion` contain the NetWare service (or OS) version information, not the NetWare product version. To get these values, type version on the server and look at the returned Server Version value. The following table lists common NetWare products and their corresponding values:

| NetWare Product | `netwareVersion` value | `netwareSubVersion` value |
|---|---|---|
| NetWare 4.11 | 4 | 11 |
| NetWare 5.1 | 5 | 0 |
| NetWare 6 | 5 | 60 |

The following fields in this structure contain different information depending on the version of CLIB used in calling the function, whether the call is made to a local or remote server, and the version of the server that is queried:

- `maxConnectionsSupported`
- `connectionsInUse`
- `peakConnectionsUsed`

The following describes the contents of the `maxConnectionsSupported` field under different circumstances.

| | |
|---|---|
| Calling a remote pre-4.0 server | Contains the size of the workstation connection table, which corresponds to the number of workstation connections the server supports (for example, 250 for 250-User NetWare). |
| Calling the local pre-4.0 server | Contains the same value as `maxConnectionsSupported` when calling a remote pre-4.0 server (above) **plus** the number of NLM connections that the server supports. If the number of NLM connections supports is 100, then a 250-user server would give a value of 350 in this field. |

| | |
|---|---|
| Calling a 4.x server, local or remote | Contains the maximum number of connections (licensed and unlicensed) that have been simultaneously attached, or are currently attached to the server. This value does not decrease. For example, if 15 workstations have been simultaneously logged in, but only three workstations are logged in when GetServerInformation is called, maxConnectionsSupported will contain a value of 15.<br><br>`maxConnectionsSupported` does not contain the maximum number of licensed connections that are possible for the 4.x server. This is because 4.x servers use licensing technology to manage users and now allow an unlimited amount of unlicensed connections. Therefore, this value is not a static value. |

The following describes the contents of the `connectionsInUse` field under different circumstances.

| | |
|---|---|
| Calling a pre-4.0 server, local or remote | Contains the number of workstation connections currently attached to the server. The number does not include internal NLM connections, nor does it reflect whether the connections are logged-in (licensed) or not. |
| Calling a 4.x server, local or remote | Contains the number of licensed workstation or internal NLM connections currently attached to the server. The number does not include connections which are not licensed. |

The following describes the contents of the `peakConnectionsUsed` field under different circumstances.

| | |
|---|---|
| Calling a pre-4.0 server, local or remote | Contains the maximum number of workstation connections that are simultaneously attached to the server. The number does not include internal NLM connections, nor does it reflect whether the connections are logged-in (licensed) or not. |
| Calling a 4.x server, local or remote | Contains the maximum number of licensed workstation or internal NLM connections that are simultaneously attached to the server. The number does not include connections which are not licensed. |

The connection licensing of 4.x brought on a fundamental change in the meaning of the above fields. CLIB version 4.0 and later supports additional "statistical service" functions to provide NetWare 4.x specific information about connections. (See the "SS" functions, such as SSGetActiveConnListByType, SSGetFileServerInfo, SSGetOSVersionInfo, SSGetUserInfo). SSGetOSVersionInfo returns a `maxNumOfConn` field, which is the size of the connection table, and another field `maxNumOfUsers`, which is the maximum number of licensed connections except on NetWare versions using Novell Licensing Services (NLS).

Using these functions in combination with the information returned from GetServerInformation should give you all you need to know about the connections on NetWare 4.x servers, with the following important exception.

**IMPORTANT:** Under the NetWare 5.x or 6.x OS or any other NetWare version that uses Novell Licensing Services (NLS), the `maxNumOfUsers` value returned by SSGetOSVersionInfo is not always equal to the maximum nubmer of licensed connections. For this reason, Novell strongly discourages tying the value of `maxNumOfUsers` to the licensing of any product.

## See Also

GetFileServerDescriptionStrings (page 439)

## GetServerInformation Example

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

main()
{
   int               ccode, structSize;
   FILE_SERV_INFO   sbuf;
   printf ("\n\n");
   structSize = 128;
   ccode = GetServerInformation (structSize, &sbuf);
   if (ccode == 0)
   {
      printf ("ServerName: %s\n", sbuf.serverName);
      printf ("NetwareVersion: %d\n", sbuf.netwareVersion);
      printf ("NetwareSubVersion: %d\n", sbuf.netwareSubVersion);
      printf ("Max Connections Supported: %d\n",
               sbuf.maxConnectionsSupported);
      printf ("Connections In Use:%d\n", sbuf.connectionsInUse);
      printf ("Max Volumes Supported: %d\n",
               sbuf.maxVolumesSupported);
      printf ("Revision Level: %d\n", sbuf.revisionLevel);
      printf ("SFT Level: %d\n", sbuf.SFTLevel);
      printf ("TTS Level: %d\n", sbuf.TTSLevel);
      printf ("Peak Connections Used: %d\n",
               sbuf.peakConnectionsUsed);
      printf ("Accounting Version: %d\n", sbuf.accountingVersion);
      printf ("VAP Version: %d\n", sbuf.VAPversion);
      printf ("Queing version: %d\n", sbuf.queingVersion);
      printf ("Print Server Version: %d\n",
               sbuf.printServerVersion);
      printf ("Virtual Console Version: %d\n",
               sbuf.virtualConsoleVersion);
      printf ("Security Restriction Level: %d\n",
               sbuf.securityRestrictionLevel);
      printf ("Internet Bridge Support: %d\n",
               sbuf.internetBridgeSupport);
      printf ("CLIB Major Version: %d\n", sbuf.CLibMajorVersion);
      printf ("CLIB Minor Version: %d\n", sbuf.CLibMinorVersion);
      printf ("CLIB Revision: %d\n", sbuf.CLibRevision);
   }
   else
      printf ("ccode = %d\n", ccode);
}
```

# GetServerMemorySize

Returns the amount of memory (RAM) on the server. (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*).)

**Local Servers:** nonblocking

**Remote Servers:** N/A

**NetWare Server:** 3.12E, 3.2, 4.01C, 4.1

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

LONG GetServerMemorySize (void);
```

## Return Values

The amount of memory (in bytes) known to the NetWare server, or EFAILURE.

## Remarks

GetServerMemorySize returns EFAILURE if it is called in the MS Engine of an SFT III server.

## GetServerMemorySize Example

```
#include <stdio.h>
#include <stdlib.h>
#include <nlm\nit\nwenvrn.h>
#include <process.h>
#include <conio.h>

main()
{
   printf("Server memory: %i\n",
   GetServerMemorySize());
}
```

# GetServerUtilization

Returns the server utilization value that can be seen from MONITOR.NLM

**Local Servers:** nonblocking

**Remote Servers:** N/A

**NetWare Server:** 3.12E, 3.2, 4.01C, 4.1

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

LONG GetServerUtilization (void);
```

## Return Values

The utilization value as computed by MONITOR.NLM.

## Remarks

The utilization of a NetWare server is a value that can be used to make a measurement of how busy a server is. This value is computed based upon how many processes running idle loops have run on the server. The value returned by GetServerUtilization is a measurement at a single instance in time.

Since the value returned by GetServerUtilization is the utilization at a single instance in time, determining a complete picture of a server's utilization cannot be determined by one call to GetServerUtilization. A more meaningful measurement of how the server is being utilized is determined by computing an average value of the utilization and determining the deviation from the average.

## GetServerUtilization Example

```
<include stdlib.h>
<include nwenvrn.h>
<include process.h>
<include conio.h>

void main(void)
{
   while(TRUE)
   {
      printf("\nUtilization: %i\n",
      GetServerUtilization());
      ThreadSwitchWithDelay();
   }
}
```

# SendConsoleBroadcast

Sends a message to a list of connections (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWSendConsoleBroadcast)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int SendConsoleBroadcast (
   char   *message,
   WORD   connectionCount,
   WORD   *connectionList);
```

## Parameters

**message**

(IN) Specifies the string containing the message to send (maximum 60 characters, including the NULL terminator).

**connectionCount**

(IN) Specifies the number of logical connections in the `connectionList` (0 = broadcast to all workstations).

**connectionList**

(IN) Contains a list of logical connection numbers to receive the message.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

Messages do not reach a workstation that has disabled broadcasts.

The requesting workstation must have console Operator rights.

## See Also

SendBroadcastMessage

## SendConsoleBroadcast Example

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

main()
{
   int   i, ccode;
   char  message[60];
   WORD  connectionList[100];
   WORD  connectionCount;
   char  buf[4];
   printf ("\n\n");
   printf ("Enter number of messages to send:   ");
   gets (buf);
   connectionCount = (WORD) atoi (buf);
   for (i=0; i<connectionCount; i++)
   {
      printf ("\nEnter connection number %d: ", i+1);
      gets (buf);
      connectionList[i] = (WORD) atoi (buf);
   }
   printf ("\nEnter message to be sent:\n");
   printf ("--> ");
   gets (message);
   ccode = SendConsoleBroadcast (message,
                                 connectionCount,
                                 connectionList);
   if (ccode == 0)
      printf("--- SUCCESSFUL ---\n\n");
   if (ccode == 198)
      printf("-- NO_CONSOLE_RIGHTS --\n\n");
}
```

# SetFileServerDateAndTime

Sets the date and time of the server (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWSetFileServerDateAndTime (page 169))

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

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

int SetFileServerDateAndTime (
   WORD   year,
   WORD   month,
   WORD   day,
   WORD   hour,
   WORD   minute,
   WORD   second);
```

## Parameters

**date and time**

(IN) These parameters set the server's date and time.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 198 | (0xC6) | ERR_NO_CONSOLE_RIGHTS |

## Remarks

The date and time parameters accept information in the following format:

| Decimal | Corresponds To | Values | Explanation |
|---|---|---|---|
| 0 | Year | 0 to 179 | 0-79 and 100-179 correspond to the years 2000-2079. 80-99 correspond to the years 1980-1999. |

| Decimal | Corresponds To | Values | Explanation |
|---|---|---|---|
| 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 | A value of 0 = Sunday,1 = Monday, 2 = Tuesday, and so on |

If an invalid value, such as 250 for the month, is specified, an error will not be returned. The server will be set to an undefined but valid date and time.

The date and time are not synchronized across the network (between servers).

The requesting workstation must have console Operator or Supervisor rights.

**NOTE:** If you are running time synchronization (NetWare 4.x) this function behaves differently depending on the server whose time you change:

If you change the time of the Reference Server, the time for the network is eventually changed.

If you change the time of a Primary Server, its time is eventually synchronized to the network time.

If you change the time of a Secondary Server, its time is eventually synchronized to the network time.

## See Also

GetFileServerDateAndTime (page 437)

## SetFileServerDateAndTime Example

```
#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <nwtypes.h>
#include <nlm\nit\nwenvrn.h>

main()
{
   int rc;
   WORD year, month, day, hour, minute, second;
   printf("enter year, month, day, hour, minute, second:\n");
   scanf("%hu,%hu,%hu,%hu,%hu,%hu", &year, &month, &day, &hour,
      &minute, &second);
   rc = SetFileServerDateAndTime(year, month, day, hour,
      minute, second);
```

```
    printf("rc = %d",rc);
}
```

## TTSGetStats (Obsolete-moved from .h file 4/99)

Obsolete—last documented in Release 15 of the Limited Support NetWare SDK.

# 11.3 SSGetA*-SSGetK* Functions

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

# SSGetActiveConnListByType

Returns a list of active connection numbers of a given connection type. For cross-platform functionality, use NWGetActiveConnListByType (page 45).

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetActiveConnListByType (
   LONG    startConnNumber,
   LONG    connType,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startConnNumber`**

(IN) Specifies the connection number to start with.

**`connType`**

(IN) Specifies the type of connection to return information for.

**`buffer`**

(IN/OUT) Points to a buffer which receives a list of connections.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetActiveConnListByTypeStructure.

## Return Values

ESUCCESS or NetWare® errors.

## Remarks

This function returns a connection list of all connections of a given type on the currently connected server. The `connType` parameter can have one of the following values:

| | |
|---|---|
| 1 | (Included for CLIB backwards compatibility) |

| | |
|---|---|
| 2 | NCP_CONNECTION_TYPE |
| 3 | NLM_CONNECTION_TYPE |
| 4 | AFP_CONNECTION_TYPE |
| 5 | FTAM_CONNECTION_TYPE |
| 6 | ANCP_CONNECTION_TYPE |

This function returns a GetActiveConnListByTypeStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetActiveConnListByTypeStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   BYTE   ActiveConnBitList[512];
}GetActiveConnListByTypeStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `ActiveConnBitList` field indicates active connections. An array of 512 bytes is returned where a bit is set for each active connection. The connection number is determined by its position in the array.

# SSGetActiveLANBoardList

Returns information about the active LAN boards on a server. For cross-platform functionality, use NWGetActiveLANBoardList (page 47) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetActiveLANBoardList (
   LONG    startNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startNumber`**

(IN) Specifies the number of LAN board to start with.

**`buffer`**

(IN/OUT) Points to a buffer which receives a list of LAN boards.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetActiveLANBoardListStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetActiveLANBoardListStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   maxNumOfLANs;
   LONG   itemsCount;
```

```
   LONG   boardNumbers;
}GetActiveLANBoardListStructure;
```

The currentServerTime field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The vConsoleVersion field contains the console version number. vConsoleVersion and vConsoleRevision track packet format.

The vConsoleRevision field contains the console version revision number.

The maxNumOfLANs field contains the total number of LAN boards.

The itemsCount field contains the number of LAN boards returned by this call to SSGetActiveLANBoardList. To retrieve the rest of the board numbers, call this function again, using the total number of items returned by all previous calls to SSGetActiveLANBoardList plus 1 as startNumber.

The boardNumbers field contains the first LAN board number. This number is followed in the buffer by board numbers for each LAN board.

# SSGetActiveProtocolStacks

Returns protocol stack information. For cross-platform functionality, use NWGetActiveProtocolStacks (page 49) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetActiveProtocolStacks (
   LONG    startNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startNumber`**

(IN) Specifies the number to start with if this function is being called iteratively. On the first call, this parameter should be 0. On subsequent calls, use the number of stacks retrieved.

**`buffer`**

(IN/OUT) Points to a buffer which receives protocol stack information.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetActiveProtocolStackStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetActiveProtocolStackStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   maxNumberOfStacks;
```

```
   LONG   stackCount;
   ProtocolStackInfo stackInfo;
}GetActiveProtocolStackStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `numberOfSegments` field contains the number of volume segments on the volume.

The `maxNumberOfStacks` field contains the total number of protocol stacks.

The `stackCount` field contains the number of ProtocolStackInfo structures in the buffer.

The `stackInfo` field contains the first ProtocolStackInfo structure in the buffer.

The ProtocolStackInfo structure is defined in NWSERVST.H as follows:

```
typedef struct ProtocolStackInfo
{
   LONG   stackNumber;
   BYTE   stackName[16];
}ProtocolStackInfo;
```

The `stackNumber` field contains the protocol number.

The `stackName` field contains a length-preceeded string that represents the name of the protocol stack.

# SSGetCacheInfo

Returns information about a server's cache buffers. For cross-platform functionality, use NWGetCacheInfo (page 51) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetCacheInfo (
   BYTE   *buffer,
   LONG   bufferLen);
```

## Parameters

**`buffer`**

(IN/OUT) Points to a buffer which receives cache buffer information structures.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetCacheInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns cache buffer information for the current connection.

This function returns the GetCacheInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetCacheInfoStructure
{
   LONG                  currentServerTime;
   BYTE                  VConsoleVersion;
   BYTE                  VConsoleRevision;
   WORD                  reserved;
   LONG                  CacheCntrs[26];
   CacheMemoryCounters   MemoryCntrs;
   CacheTrendCounters    TrendCntrs;
```

```
   CacheInformation      CacheInfo;
}GetCacheInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `VConsoleVersion` field contains the console version number. `VConsoleVersion` and `VConsoleRevision` track packet format.

The `VConsoleRevision` field contains the console version revision number.

The `CacheCntrs` field contains an array of counters that are used by Novell® for debugging.

The `MemoryCntrs`, `TrendCntrs`, and `CacheInfo` fields contain structures which hold cache buffer information.

The CacheMemoryCounters structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   OriginalNumberOfCacheBuffers;
   LONG   CurrentNumberOfCacheBuffers;
   LONG   CacheDirtyBlockThreshold;
   LONG   debugCounters[7];
} CacheMemoryCounters;
```

The `OriginalNumberOfCacheBuffers` field contains the number of cache buffers that existed when the server was brought up.

The `CurrentNumberOfCacheBuffers` field contains the number of cache buffers presently on the server.

The `CacheDirtyBlockThreshold` field contains the maximum number o cache blocks allowed to be dirty simultaneously.

The `debugCounters` field contains an array of counters that are used by Novell for debugging.

The CacheTrendCounters structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   NumOfCacheChecks;
   LONG   NumOfCacheHits;
   LONG   debugCounters[7];
   LONG   LRUSittingTime;
} CacheTrendCounters;
```

The `NumOfCacheChecks` field contains the total number of times any block in the cache was looked at since the server was brought up.

The `NumOfCacheHits` field contains the number of times cache requests were serviced from existing cache blocks.

The `debugCounters` field contains an array of counters that are used by Novell for debugging.

The `LRUSittingTime` field contains the time in ticks that the oldest cache block has been available (sitting in the LRU list).

The CacheInformation structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG MaximumByteCount;
   LONG   MinimumNumberOfCacheBuffers;
   LONG   MinimumCacheReportThreshold;
   LONG   AllocateWaitingCount;
   LONG   NDirtyBlocks;
   LONG   CacheDirtyWaitTime;
   LONG   CacheMaximumConcurrentWrites;
   LONG   MaximumDirtyTime;
   LONG   NumberOfDirectoryCacheBuffers;
   BYTE   CacheByteToBlockShiftFactor;
} CacheInformation;
```

The `MaximumByteCount` field contains the maximum length (in bytes) of a cache block.

The `MinimumNumberOfCacheBuffers` field contains the minimum number of cache buffers allowed on the server. This number can be set using the SET console command. Supported values are 20 to 1000; the default value is 20.

The `MinimumCacheReportThreshold` field contains the number of cache buffers used for the report threshold. When the cache buffers reach a number equal to the minimum number of cache buffers plus this report threshold, the server sends an alarm message warning that cache buffers are getting low. This number can be set using the SET console command. Supported values are 0 to 1000; the default value is 20.

The `AllocateWaitingCount` field contains the number of processes waiting to allocate a cache block.

The `NDirtyBlocks` field contains the number of dirty blocks waiting to write to disk.

The `CacheDirtyWaitTime` field contains the maximum wait before a write request is written to disk. This value can be set using the SET console command. Supported values are 0.1 seconds to 10 seconds; the default value is 3.3 seconds.

The `CacheMaximumConcurrentWrites` field contains the maximum number of write requests for changed file data that can be put in the elevator before the disk head begins a sweep across the disk. This value can be set using the SET console command. Supported values are 10 to 100; the default value is 50.

The `MaximumDirtyTime` field contains the longest time (in ticks) that a dirty block has waited before it was written to disk since the server was brought up.

The `NumberOfDirectoryCacheBuffers` field contains the number of directory cache buffers on the server.

The `CacheByteToBlockShiftFactor` field contains the factor used to determine the block size. Block size is calculated by the following equation, where `n` is the shift factor:

block size = $2^{n}$ bytes

# SSGetCPUInfo

Returns information about the server CPU. For cross-platform functionality, use NWGetCPUInfo (page 53) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetCPUInfo (
   LONG    CPUNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**CPUNumber**

(IN) Specifies the number of the CPU to obtain information for (currently this number is always 0).

**buffer**

(IN/OUT) Points to a buffer which receives CPU information.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetCPUInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetCPUInfoStructure
{
   LONG             currentServerTime;
   BYTE             vConsoleVersion;
   BYTE             vConsoleRevision;
   WORD             reserved;
   CPUInformation   CPUInfo;
```

```
   BYTE              variableStringsStart;
}GetCPUInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `CPUInfo` field contains a CPUInformation structure.

The `variableStringsStart` field contains the CPU string, the NPX present string, and the Bus string.

The `CPUInfo` field contains a CPUInformation structure. This structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   numberOfCPUs;
   LONG   PageTableOwnerFlag;
   LONG   CPUType;
   LONG   CoProcessorFlag;
   LONG   BusType;
   LONG   IOEngineFlag;
   LONG   FSEngineFlag;
   LONG   NonDedFlag;
} CPUInformation;
```

The `numberOfCPUs` field contains the number of registered CPUs in the server.

The `PageTableOwnerFlag` field indicates whether the NetWare OS owns the page table. Values for this flag include the following:

0-The NetWare OS owns the page tables
1-The NetWare OS does not own the page tables (currently, only when running with OS/2)

The `CPUType` field contains the CPU type.

The `CoProcessorFlag` field indicates whether a coprocessor is present.

The `BusType` field indicates the type of bus used:

| | |
|---|---|
| 0x01 | microchannel |
| 0x02 | EISA |
| 0x04 | PCI |
| 0x08 | PCMCIA |
| 0x10 | ISA |

More than one of the bit field can be set. Thus, for example, if PCI and ISA are both supported, the value in the `BusType` field is 0x14 (decimal 20).

The `IOEngineFlag` field indicates whether the IO engine is installed (TRUE = installed).

The `FSEngineFlag` field indicates whether the file system engine is installed (TRUE = installed).

The `NonDedFlag` currently is not supported. In the future, this field will indicate whether the CPU is dedicated to the NetWare OS. This flag is set when the CPU is nondedicated (currently, only when running OS/2).

The `PageTableOwnerFlag`, defined above, can be used to determine whether an NLM™ application is running under NetWare for OS/2 because this flag is 0 only under those circumstances.

# SSGetDirCacheInfo

Returns information about the directory cache of a server. For cross-platform functionality, use NWGetDirCacheInfo (page 55) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetDirCacheInfo (
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`buffer`**

(IN/OUT) Points to a buffer which receives directory cache information.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetDirCacheInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns information for the current connection. A GetDirCacheInfoStructure is returned in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetDirCacheInfoStructure
{
   LONG                        currentServerTime;
   BYTE                        vConsoleVersion;
   BYTE                        vConsoleRevision;
   WORD                        reserved;
   DirectoryCacheInformation   dirCacheInfo;
}GetDirCacheInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The vConsoleVersion field contains the console version number. vConsoleVersion and vConsoleRevision track packet format.

The vConsoleRevision field contains the console version revision number.

The dirCacheInfo field contains a DirectoryCacheInformation structure.

The DirectoryCacheInformation structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   MinimumTimeSinceFileDelete;
   LONG   AbsMinimumTimeSinceFileDelete;
   LONG   MinimumNumberOfDirCacheBuffers;
   LONG   MaximumNumberOfDirCacheBuffers;
   LONG   NumberOfDirectoryCacheBuffers;
   LONG   DCMinimumNonReferencedTime;
   LONG   DCWaitTimeBeforeNewBuffer;
   LONG   DCMaximumConcurrentWrites;
   LONG   DCDirtyWaitTime;
   LONG   debugCounters[4];
   LONG   PercentOfVolumeUsedByDirs;
} DirectoryCacheInformation;
```

The MinimumTimeSinceFileDelete field contains the minimum time (in ticks) between the deletion of a file and when it can be purged.

The AbsMinimumTimeSinceFileDelete field contains the minimum time (in ticks) between the deletion of a file and when it can be purged when the system has no available blocks.

The MinimumNumberOfDirCacheBuffers field contains the minimum number of directory cache buffers that can be allocated on the server. This number can be set using the SET console command. Supported values are 10 to 2000; the default value is 20.

The MaximumNumberOfDirCacheBuffers field contains the maximum number of directory cache buffers that can be allocated on the server. This number can be set using the SET console command. Supported values are 20 to 4000; the default value is 500.

The NumberOfDirectoryCacheBuffers field contains the current number of directory cache buffers on the server.

The DCMinimumNonReferencedTime field contains the time (in ticks) that must elapse between the last reference of a directory buffer and the time it is reused. This value can be set using the SET console command. Supported values are 1 second to 5 minutes; the default value is 5.5 seconds.

The DCWaitTimeBeforeNewBuffer field contains the time (in ticks) that must elapse before an additional directory cache buffer is allocated. This value can be set using the SET console command. Supported values are 0.5 seconds to 2 minutes; the default value is 2.2 seconds.

The DCMaximumConcurrentWrites field contains the maximum number of write requests from directory cache buffers that can be put in the elevator before they are written to disk. This value can be set using the SET console command. Supported values are 5 to 50; the default value is 10.

The DCDirtyWaitTime field contains the maximum time (in ticks) that the server can wait before writing dirty cache buffers to disk. This value can be set using the SET console command. Supported values are 0 to 10 seconds; the default value is 0.5 seconds.

The `debugCounters` field contains an array of counters used by Novell for debugging.

The `PercentOfVolumeUsedByDirs` field contains the maximum percentage of a volume that can be used by directories. This value can be set using the SET console command. Supported values are 5 to 50; the default value is 13.

# SSGetFileServerInfo

Returns information about a server. For cross-platform functionality, use NWGetFileServerInfo (page 65) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetFileServerInfo (
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`buffer`**

(IN/OUT) Points to a buffer which receives server information structures.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetFileServerInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns server information for the current connection.

---

**NOTE:** Prior to NetWare 5.1 SP 6 and NetWare 6 SP 3, the SSGetFileServerInfo function required the IPX protocol stack to be loaded on the server. With these service packs, IP-only servers return all information except the IPX-specific information. These fields (`NCPStaInUseCnt`, `NCPPeakStaInUse`, and `numOfNCPReqs`) always return 0.

---

This function returns the GetFileServerInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetFileServerInfoStructure
{
   LONG                currentServerTime;
   BYTE                vConsoleVersion;
   BYTE                vConsoleRevision;
```

```
   WORD                reserved;
   LONG                NCPStaInUseCnt;
   LONG                NCPPeakStaInUse;
   LONG                numOfNCPReqs;
   LONG                serverUtilization;
   ServerInformation   serverInfo;
   FSCounters          fileServerCounters;
}GetFileServerInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `NCPStaInUseCnt` field contains the number of workstations connected to the server.

The `NCPPeakStaInUse` field contains the maximum number of workstations connected at one time since the server was brought up.

The `numOfNCPReqs` field contains the number of NCP™ requests received by the server since it was brought up.

The `serverUtilization` field contains the current percentage of CPU utilization for the server.

The `serverInfo` and `fileServerCounters` fields contain server statistics.

The ServerInformation structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   ReplyCanceledCount;
   LONG   WriteHeldOffCount;
   LONG   reserved1;
   LONG   InvalidRequestTypeCount;
   LONG   BeingAbortedCount;
   LONG   AlreadyDoingReAllocateCount;
   LONG   reserved2[3];
   LONG   DeAllocateStillTransmittingCount;
   LONG   StartStationErrorCount;
   LONG   InvalidSlotCount;
   LONG   BeingProcessedCount;
   LONG   ForgedPacketCount;
   LONG   StillTransmittingCount;
   LONG   ReExecuteRequestCount;
   LONG   InvalidSequenceNumberCount;
   LONG   DuplicateIsBeingSentAlreadyCount;
   LONG   SentPositiveAcknowledgeCount;
   LONG   SentADuplicateReplyCount;
   LONG   NoMemoryForStationControlCount;
   LONG   NoAvailableConnectionsCount;
   LONG   ReAllocateSlotCount;
   LONG   ReAllocateSlotCameTooSoonCount;
} ServerInformation;
```

The ReplyCanceledCount field contains the number of replies that were cancelled because the connection was reallocated while the request was being processed.

The WriteHeldOffCount field contains the number of times that writes were delayed because of a pending TTS™ transaction or cache busy condition.

The InvalidRequestTypeCount field contains the number of packets received which had an invalid request type or were received after the server was downed.

The BeingAbortedCount field contains the number of packets received for a connection that was being terminated.

The AlreadyDoingReAllocateCount field contains the number of times that a connection is requested when a connection already exists.

The DeAllocateInvalidSlotCount field contains the number of times an attempt was made to deallocate a connection slot which was not valid.

The StartStationErrorCount field contains the number of times the server was unable to allocate a connection for whatever reason.

The InvalidSlotCount field contains the number of requests received for an invalid connection slot.

The BeingProcessedCount field contains the number of times a duplicate request was received during processing of the first request.

The ForgedPacketCount field contains the number of suspicious invalid packets received. It is rarely possible to create such packets because of faulty equipment. If this number is large, it might indicate an attempt to breach network security.

The StillTransmittingCount field contains the number of times a new request is received before a reply to a previous request has been sent.

The ReExecuteRequestCount field contains the number of times the requester did not receive the reply and the request had to be reprocessed.

The InvalidSequenceNumberCount field contains the number of request packets the server received from a connection where the sequence number in the packet did not match the current sequence number or the next sequence number. (Packets with bad sequence numbers are discarded). If this number is large, it might indicate an attempt to breach network security.

The DuplicateIsBeingSentAlreadyCount field contains the number of times a duplicate reply was requested when the reply had already been sent.

The SentPositiveAcknowledgeCount field contains the number of acknowledgments sent by the server. An acknowledgment is sent when a connection repeats a request that is being serviced.

The SentADuplicateReplyCount field contains the number of request packets for which the server had to send a duplicate reply. (Duplicate replies are only sent for requests the server cannot process.)

The NoMemoryForStationControlCount field contains the number of times that the server could not allocate memory to expand the connection table for a new connection.

The NoAvailableConnectionsCount field contains the number of times there were no slots available in the connection table for a new connection.

The ReAllocateSlotCount field contains the number of times the server reallocated the same slot in the connection table for a client that logged out and then re-logged in.

The ReAllocateSlotCameTooSoonCount field contains the number of times that a request came from a client to re-log in before that client had been completely logged out.

The FSCounters structure is defined in NWSERVST.H as follows:

```
typedef struct {
   WORD   TooManyHops;
   WORD   UnknownNetwork;
   WORD   NoSpaceForService;
   WORD   NoRecieveBuffers;
   WORD   NotMyNetwork;
   LONG   NetBIOSProgatedCount;
   LONG   TotalPacketsServiced;
   LONG   TotalPacketsRouted;
} FSCounters;
```

The TooManyHops field contains the number of packets that were discarded because they had passed through more than 16 bridges without reaching their destination.

The UnknownNetwork field contains the number of packets that were discarded because their destination network was unknown to the server.

The NoSpaceForService field is always set to 0 (currently).

The NoRecieveBuffers field contains the number of times a packet was discarded because there were no buffers to receive it.

The NotMyNetwork field contains the number of packets received that were not destined for the server.

The NetBIOSPropagatedCount field contains the number of NetBIOS packets received that were sent forward.

The TotalPacketsServiced field contains the total packets received by the server.

The TotalPacketsRouted field contains the number of all packets forwarded by the server.

# SSGetFileSystemInfo

Returns information about a server's file system.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetFileSystemInfo (
   LONG   fileSystemID,
   BYTE  *buffer,
   LONG   bufferLen);
```

## Parameters

**`fileSystemID`**

(IN) Specifies the ID number of the file system for which to return information. Currently this value is always 1 (for 386).

**`buffer`**

(IN/OUT) Points to a buffer which receives file system information.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetFileSystemInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns file system information for the file system specified by `fileSystemID`. Although none of the information currently returned by this function is useful to a developer, this function has been included in the documentation for future use.

This function returns a GetFileSystemInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetFileSystemInfoStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
```

```
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   debugCounters[13];
}GetFileSystemInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `debugCounters` field contains an array of counters that deal with the background FAT update process. These counters are used by Novell for debugging.

# SSGetGarbageCollectionInfo

Returns information about garbage collection on a server. For cross-platform functionality, use NWGetGarbageCollectionInfo (page 79) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetGarbageCollectionInfo (
   BYTE   *buffer,
   LONG   bufferLen);
```

## Parameters

**buffer**

(IN/OUT) Points to a buffer which receives garbage collection information.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be the size of GetGarbageCollInfoStruc.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetGarbageCollInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetGarbageCollInfoStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   failedAllocReqCount;
   LONG   numberOfAllocs;
   LONG   noMoreMemAvlCnt;
   LONG   numOfGarbageColl;
   LONG   foundSomeMem;
```

```
   LONG    numOfChecks;
}GetGarbageCollInfoStruc;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `failedAllocReqCount` field contains the number of times memory allocation failed since the server was brought up.

The `numberOfAllocs` field contains the number of memory allocations made since the server was brought up.

The `noMoreMemAvlCnt` field contains the number of times that allocation failed because there was no memory available since the server was brought up.

The `numOfGarbageColl` field contains the number of times garbage collection was invoked since the server was brought up.

The `foundSomeMem` field contains the number of times garbage collection reclaimed memory.

The `numOfChecks` field contains the number of times garbage collection checked for memory since the server was brought up.

# SSGetIPXSPXInfo

Returns information about IPX™ / SPX™ use on a server. For cross-platform functionality, use NWGetIPXSPXInfo (page 83) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetIPXSPXInfo (
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`buffer`**

(IN/OUT) Points to a buffer which receives IPX and SPX information.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetIPXSPXInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetIPXSPXInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetIPXSPXInfoStructure
{
   LONG             currentServerTime;
   BYTE             vConsoleVersion;
   BYTE             vConsoleRevision;
   WORD             reserved;
   IPXInformation   IPXInfo;
   SPXInformation   SPXInfo;
}GetIPXSPXInfoStructure;
```

The currentServerTime field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The vConsoleVersion field contains the console version number. vConsoleVersion and vConsoleRevision track packet format.

The vConsoleRevision field contains the console version revision number.

The IPXInfo field contains a IPXInformation structure. This structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   IpxSendPacketCount;
   WORD   IpxMalformPacketCount;
   LONG   IpxGetECBRequestCount;
   LONG   IpxGetECBFailCount;
   LONG   IpxAESEventCount;
   WORD   IpxPostponedAESCount;
   WORD   IpxMaxConfiguredSocketCount;
   WORD   IpxMaxOpenSocketCount;
   WORD   IpxOpensocketFailCount;
   LONG   IpxListenECBCount;
   WORD   IpxECBCancelFailCount;
   WORD   IpxGetLocalTargetFailCount;
} IPXInformation;
```

The IpxSendPacketCount field contains the number of IPX packets sent by the server.

The IpxMalformPacketCount field contains the number of IPX packets discarded because they were malformed.

The IpxGetECBRequestCount field contains the number of ECB requests.

The IpxGetECBFailCount field contains the number of times an ECB was requested but could not be supplied.

The IpxAESEventCount field contains the number of AES events scheduled.

The IpxPostponedAESCount field contains the number of AES events that could not be scheduled and were placed in a waiting list.

The IpxMaxConfiguredSocketCount field contains the maximum number of sockets that can be open at one time.

The IpxMaxOpenSocketCount field contains the maximum number of sockets open at one time since the server was brought up.

The IpxOpensocketFailCount field contains the number of times a request to open a socket failed.

The IpxListenECBCount field contains the number of ECBs listening for a packet.

The IpxECBCancelFailCount field contains the number of ECB listens that were cancelled.

The IpxGetLocalTargetFailCount field contains the number of times that the server failed to find the target.

The SPXInfo field contains a SPXInformation structure. This structure is defined in NWSERVST.H as follows:

```
typedef struct {
   WORD   SpxMaxConnectionsCount;
   WORD   SpxMaxUsedConnections;
   WORD   SpxEstConnectionReq;
   WORD   SpxEstConnectionFail;
   WORD   SpxListenConnectReq;
   WORD   SpxListenConnectFail;
   LONG   SpxSendCount;
   LONG   SpxWindowChokeCount;
   WORD   SpxBadSendCount;
   WORD   SpxSendFailCount;
   WORD   SpxAbortedConnection;
   LONG   SpxListenPacketCount;
   WORD   SpxBadListenCount;
   LONG   SpxIncomingPacketCount;
   WORD   SpxBadInPacketCnt;
   WORD   SpxSuppressedPackCnt;
   WORD   SpxNoSesListenECBCnt;
   WORD   SpxWatchDogDestSesCnt;
} SPXInformation;
```

The SpxMaxConnectionsCount field contains the maximum number of SPX connections allowed on the server.

The SpxMaxUsedConnections field contains the maximum number of SPX connections used at one time since the server was brought up.

The SpxEstConnectionReq field contains total number of SPX connections established since the server was brought up.

The SpxEstConnectionFail field contains number of times that an attempt to establish an SPX connection failed since the server was brought up.

The SpxListenConnectReq field contains the number of requests to post a listen since the server was brought up.

The SpxListenConnectFail field contains the number of times a request to post a listen failed since the server was brought up.

The SpxSendCount field contains the number of SPX packets sent since the server was brought up.

The SpxWindowChokeCount field contains a value used internally for debugging.

The SpxBadSendCount field contains the number of bad packets sent since the server was brought up.

The SpxSendFailCount field contains the number of packets sent for which no acknowledgment was received since the server was brought up.

The SpxAbortedConnection field contains the number of times a connection was aborted since the server was brought up.

The `SpxListenPacketCount` field contains the number of times a listen was posted on a socket since the server was brought up.

The `SpxBadListenCount` field contains the number of times a listen on a socket failed for whatever reason (for example, it was cancelled) since the server was brought up.

The `SpxIncomingPacketCount` field contains the number of packets in the queue.

The `SpxBadInPacketCnt` field contains the number of bad SPX packets received since the server was brought up.

The `SpxSuppressedPackCnt` field contains the number of times a duplicate SPX packet was received.

The `SpxNoSesListenECBCnt` field contains the number of times a listen was posted on a session that was not established since the server was brought up.

The `SpxWatchDogDestSesCnt` field contains the number of times the watchdog destroyed a session since the server was brought up.

# SSGetKnownNetworksInfo

Returns information about known networks. For cross-platform functionality, use NWGetKnownNetworksInfo (page 85) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetKnownNetworksInfo (
   LONG   startNumber,
   BYTE  *buffer,
   LONG   bufferLen);
```

## Parameters

**`startNumber`**

(IN) Specifies the number to start with.

**`buffer`**

(IN/OUT) Points to a buffer which receives a information about known networks.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetKnownNetworksStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetKnownNetworksStructure
{
   LONG                    currentServerTime;
   BYTE                    vConsoleVersion;
   BYTE                    vConsoleRevision;
   WORD                    reserved;
   LONG                    numberOfEntries;
```

```
   KnownNetworksStructure  info;
}GetKnownNetworksStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `numberOfEntries` field contains the number of structures in the buffer

The `info` field contains the first KnownNetworkStructure.

The KnownNetworkStructure is defined in NWSERVST.H as follows:

```
typedef struct KnownNetworksStructure
{
   LONG   netIDNumber;
   WORD   hopsToNet;
   WORD   netStatus;
   WORD   timeToNet;
}KnownNetworksStructure;
```

The `netIDNumber` field contains the network ID number.

The `hopsToNet` field contains the number of hops to the network from the server.

The `netStatus` field indicates the status of the network.

The `timeToNet` field contains the time in ticks to the network (round-trip).

# SSGetKnownServersInfo

Returns information about known servers. For cross-platform functionality, use NWGetKnownServersInfo (page 87) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetKnownServersInfo (
   LONG    startNumber,
   LONG    serverType,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startNumber`**

(IN) Specifies the number of the server to start with.

**`serverType`**

(IN) Specifies the type of server to return information for.

**`buffer`**

(IN/OUT) Points to a buffer which receives a information about protocol configuration of the server.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetKnownServerInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetKnownServersInfoStructure
{
   LONG                    currentServerTime;
```

```
   BYTE                    vConsoleVersion;
   BYTE                    vConsoleRevision;
   WORD                    reserved;
   LONG                    numberOfEntries;
   KnownServerStructure    info;
}GetKnownServersInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `numberOfEntries` field contains the number of structures in the buffer.

The `info` field contains the first KnownServerStructure in the buffer. More follow.

```
typedef struct KnownServerStructure
{
   BYTE   serverAddress[12];
   WORD   hopCount;
   BYTE   name;
}KnownServerStructure;
```

The `serverAddress` field contains the node address of the known server.

The `hopCount` field contains the number of hops to the server.

The `name` field contains the first byte of the server name.

# 11.4 SSGetL*-SSGetN* Functions

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

# SSGetLANCommonCounters

Returns common statistics for a LAN board. For cross-platform functionality, use NWGetLANCommonCountersInfo (page 89) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetLANCommonCounters (
   LONG    boardNumber,
   LONG    blockNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**boardNumber**

(IN) Specifies the board number of the LAN board for which you want information.

**blockNumber**

(IN) Specifies the block number to start with. On the first call to this function this value should be 0. On subsequent calls this value should be the value that is returned in the `nextCntBlock` field of the GetLANCommonCountersStructure returned by this function.

**buffer**

(IN/OUT) Points to a buffer which receives a GetLANCommonCountersStructure.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be the size of GetLANCommonCountersStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetLANCommonCountersStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetLANCommonCountersStructure
{
   LONG                 currentServerTime;
   BYTE                 vConsoleVersion;
   BYTE                 vConsoleRevision;
   BYTE                 statMajorVersion;
   BYTE                 statMinorVersion;
   LONG                 totalCommonCnts;
   LONG                 totalCntBlocks;
   LONG                 customCounters;
   LONG                 nextCntBlock;
   CommonLANStructure   info;
}GetLANCommonCountersStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `statMajorVersion` field contains the major version number of the generic portion of the statistics table (defined by Novell).

The `statMinorVersion` field contains the minor version number of the generic portion of the statistics table (defined by Novell).

The `totalCommonCnts` field contains the number of LAN common counters.

The `totalCntBlocks` field contains the total number of blocks used by LAN common counters for the specified LAN board.

The `customCounters` field contains the number of LAN custom counters.

The `nextCntBlock` field contains the value to be passed in `blockNumber` on the next call to this function. When 0 is returned in this field, all common counters have been returned.

The `info` field contains a CommonLANStructure.

The CommonLANStructure is defined in NWSERVST.H as follows:

```
typedef struct CommonLANStructure
{
   LONG   notSupportedMask;
   LONG   TotalTxPacketCount;
   LONG   TotalRxPacketCount;
   LONG   NoECBAvailableCount;
   LONG   PacketTxTooBigCount;
   LONG   PacketTxTooSmallCount;
   LONG   PacketRxOverflowCount;
   LONG   PacketRxTooBigCount;
   LONG   PacketRxTooSmallCount;
   LONG   PacketTxMiscErrorCount;
   LONG   PacketRxMiscErrorCount;
```

```
   LONG   RetryTxCount;
   LONG   ChecksumErrorCount;
   LONG   HardwareRxMismatchCount;
}CommonLANStructure;
```

The `notSupportedMask` field indicates if the counter is supported. If the bit is 0, the counter is supported; if it is 1, the counter is not supported.

The `TotalTxPacketCount` field contains the total number of packets transmitted by the LAN board.

The `TotalRxPacketCount` field contains the total number of packets that were received by the LAN board.

The `NoECBAvailableCount` field contains the number of times the LAN board failed to get a receive ECB.

The `PacketTxTooBigCount` field contains the number of times the send packet was too big for this LAN board to send.

The `PacketTxTooSmallCount` field contains the number of times the send packet was too small for this LAN board to send.

The `PacketRxOverflowCount` field contains the number of times the LAN board's receive buffers overflowed.

The `PacketRxTooBigCount` field contains the number of times this LAN board could not receive a packet because the packet was too big.

The `PacketRxTooSmallCount` field contains the number of times this LAN board could not receive a packet because the packet was too small.

The `PacketTxMiscErrorCount` field contains the number of times any kind of transmit error occurred for the LAN board.

The `PacketRxMiscErrorCount` field contains the number of times any kind of receive error occurred for the LAN board.

The `RetryTxCount` field contains the number of times the LAN board retried a transmit because of failure.

The `ChecksumErrorCount` field contains the number of times a checksum error occurred for the LAN board.

The `HardwareRxMismatchCount` field contains a counter which can be incremented when a packet is received which does not pass length consistency checks (currently used only by the Ethernet TSU).

# SSGetLANConfiguration

Returns information about the configuration of LAN drivers on a server. For cross-platform functionality, use NWGetLANConfigInfo (page 91) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetLANConfiguration (
   LONG    boardNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**boardNumber**

(IN) Specifies the number (1-255) of the logical LAN board to obtain information for. The value must not be zero.

**buffer**

(IN/OUT) Points to a buffer which receives a LAN driver configuration information.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be the size of GetLANConfigInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetLANConfigInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetLANConfigInfoStructure
{
   LONG                    currentServerTime;
   BYTE                    vConsoleVersion;
   BYTE                    vConsoleRevision;
   WORD                    reserved;
```

```
   DriverConfigStructure   LANConfig;
}GetLANConfigInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `LANConfig` field contains a DriverConfigStructure.

The DriverConfigStructure is defined in NWSERVST.H as follows:

```
typedef struct {
   BYTE   DriverCFG_MajorVersion;
   BYTE   DriverCFG_MinorVersion;
   BYTE   DriverNodeAddress[6];
   WORD   DriverModeFlags;
   WORD   DriverBoardNumber;
   WORD   DriverBoardInstance;
   LONG   DriverMaximumSize;
   LONG   DriverMaxRecvSize;
   LONG   DriverRecvSize;
   LONG   DriverCardName;
   LONG   DriverShortName;
   LONG   DriverMediaType;
   WORD   DriverCardID;
   WORD   DriverMediaID;
   WORD   DriverTransportTime;
   BYTE   DriverReserved[16];
   BYTE   DriverMajorVersion;
   BYTE   DriverMinorVersion;
   WORD   DriverFlags;
   WORD   DriverSendRetries;
   LONG   DriverLink;
   WORD   DriverSharingFlags;
   WORD   DriverSlot;
   WORD   DriverIOPortsAndLengths[4];
   LONG   DriverMemoryDecode0;
   WORD   DriverLength0;
   LONG   DriverMemoryDecode1;
   WORD   DriverLength1;
   BYTE   DriverInterrupt[2];
   BYTE   DriverDMAUsage[2];
   LONG   DriverResourceTag;
   LONG   DriverConfig;
   LONG   DriverCommandString;
   BYTE   DriverLogicalName[18];
   LONG   DriverLinearMemory[2];
   WORD   DriverChannelNumber;
   BYTE   DriverIOReserved[6];
} DriverConfigStructure;
```

The DriverCFG_MajorVersion field contains the Novell defined major version number of the configuration table.

The DriverCFG_MinorVersion field contains the Novell defined minor version of the configuration table.

The DriverNodeAddress field contains the node address of the LAN board.

The DriverModeFlags field contains the modes supported by the driver. The bits are defined as follows:

| | |
|---|---|
| 0 | Set to 1. At one time this bit indicated whether the driver was a real or dummy driver. |
| 1 | Set if the driver uses DMA. However, the buffers are not guaranteed to be on a 64K boundary. |
| 2 | Set only if bit 4 is set. This bit tells routers to only pass router table changes when they occur, rather than forwarding all RIP and SAP packets. |
| 3 | Set if the driver supports multicasting. |
| 4 | "Point-to-point bit": Set if the driver can bind with a protocol stack without providing a network number. |
| 6 | Set if the driver supports raw sends. If the LSL hands a "raw" packet to an MLID, the MLID should send the packet without prepending any hardware header (the protocol stack prepares the packet). |
| 10 | Set if the HSM can handle framented RCBs. |
| 13 | Set if the HSM can handle promiscuous RCBs. |
| 14-15 | Bits 14 and 15 combine to describe the DriverNodeAddress (see the following table). |

Bits 14 and 15 in DriverModeFlags describe the format of DriverNodeAddress. The combinations are as follow:

| | |
|---|---|
| 00 | Format is unspecified-The node address is assumed to be in the physical layer’s native format |
| 01 | This is an illegal combination and should not occur |
| 10 | DriverNodeAddress is canonical |
| 11 | DriverNodeAddress is non-canonical |

The DriverBoardNumber field contains the logical board number (1 - 255) assigned to the LAN board by the L ink Support Layer™ ( LSL™) software.

**WARNING:** If the value of DriverBoardNumber is zero at the time SSGetLANConfugration is called, the function never returns.

The DriverBoardInstance field contains the number of the physical card that the logical board is using. If a driver is driving one physical card, all the logical boards using this card would have a value of 1 in this field. If a second physical card is added, all the logical boards using the second physical card would have a value of 2 in this field.

The `DriverMaximumSize` field contains the maximum send or receive packet size (in bytes) that the LAN board can transmit or receive.

The `DriverMaxRecvSize` field contains the maximum packet size (or best size) in bytes that the LAN board can receive.

The `DriverRecvSize` field contains the maximum packet size (in bytes) a protocol stack can send or receive using this board.

The `DriverCardName` field contains a pointer to a length-preceded, zero-terminated ASCII description string that is contained in the OSDATA segment and is similar to the description string in the definition table. For example, "NE2000 ETHERNET Driver".

The `DriverShortName` field contains a pointer to a length-preceded, zero-terminated ASCII string that describes the LAN board in eight bytes or less, such as "NE2000".

The `DriverMediaType` field contains a pointer to a length-preceded, zero-terminated string that describes the frame type of the Multiple Link Interface Driver™ ( MLID™) software (for example, "ETHERNET_802.3"). The Independent Manufacturer Support Program (IMSP) assigns strings for the frame type. The following table summarizes frame types currently defined by Novell with their corresponding protocol ID numbers for IPX.

| Frame ID | Frame Type String | Protocol ID on IPX/ SPX | Description |
|---|---|---|---|
| 0 | VIRTUAL_LAN | 00h | For use where no Frame ID/MAC envelope is necessary |
| 1 | LOCALTALK | 00h | Apple LocalTalk* frame |
| 2 | ETHERNET_II | 8137h | Ethernet using a DEC* Ethernet II envelope |
| 3 | ETHERNET_802.2 | E0h | Ethernet (802.3) using an 802.2 envelope |
| 4 | TOKEN-RING | E0h | Token ring (802.5) using an 802.2 envelope |
| 5 | ETHERNET_802.3 | 00h | IPX 802.3 raw encapsulation |
| 6 | 802.4 | N/A | Token-passing bus envelope |
| 7 | NOVELL_PCN2 | 1111h | Novell's IBM* PC Network II envelope |
| 8 | GNET | E0h | Gateway's GNET frame envelope |
| 9 | PRONET-10 | N/A | Proteon's* ProNET* I/O frame envelope |
| 10 | ETHERNET_SNAP | 8137h | Ethernet (802.3) using an 802.2 envelope with SNAP |
| 11 | TOKEN-RING_SNAP | 8137h | Token ring (802.5) using an 802.2 envelope with SNAP |
| 12 | LANPAC_II | N/A | Racore's frame envelope |
| 13 | ISDN | N/A | Integrated Services Digital Network |
| 14 | NOVELL_RX-NET | FAh | Novell's ArcNet 68™ envelope |
| 15 | IBM_PCN2_802.2 | E0h | IBM PCN2 using 802.2 envelope |
| 16 | IBM_PCN2_SNAP | 8137h | IBM PCN2 using 802.2 with SNAP envelope |

| Frame ID | Frame Type String | Protocol ID on IPX/ SPX | Description |
|---|---|---|---|
| 17 | OMNINET/4 | N/A | Corvus's frame envelope |
| 18 | 3270_COAXA | N/A | Harris Adacom's frame envelope |
| 19 | IP | N/A | IP Tunnel frame envelope |
| 20 | FDDI_802.2 | E0h | FDDI (802.7) using an 802.2 envelope |
| 21 | IVDLAN_802.9 | N/A | Commtex, Inc.'s frame envelope |
| 22 | DATACO_OSI | N/A | Dataco's frame envelope |
| 23 | FDDI_SNAP | 8137h | FDDI (802.7) using 802.2 with a SNAP envelope |
| 24 | IBM_SDLC | N/A | SDLC tunnel envelope |
| 25 | PCO_FDDITP | N/A | PC Office frame envelope |
| 26 | WAIDNET | N/A | Hypercommunications |
| 27 | SLIP | N/A | Novell frame envelope |
| 28 | PPP | N/A | Novell frame envelope |

The `DriverCardID` field contains the number assigned to the LAN board by the IMSP.

The `DriverMediaID` field contains a number identifying the link-level envelope (frame ID) used by the MLID. The above table lists defined ID numbers.

The `DriverTransportTime` field contains the time (in ticks) that it takes for the LAN board to transmit a 576-byte packet.

The `DriverReserved` field is reserved for future use (currently set to 0).

The `DriverMajorVersion` field contains the major version number of the MLID.

The `DriverMinorVersion` field contains the minor version number of the MLID.

The `DriverFlags` field contains a bit map indicating the architecture supported by the MLID:

0 Supports an EISA board
1 Supports an ISA board
2 Supports an MCA board
8 Supports HUB management
9-10 Bits 9 and 10 combine to indicate different support mechanisms for multicast filtering and multicast formats (see the following table)

Bits 9 and 10 in `DriverFlags` indicate different support mechanisms for multicast filtering and format. The combinations are as follow:

| | |
|---|---|
| 00 | The method used for group addressing support defaults to that of the LAN medium. For example, for Ethernet it is the Hash Table; for Token Ring it is the Functional Address. |
| 01 | This is an illegal combination and should not occur |

| | |
|---|---|
| 10 | Group addressing is supported by a specialized adapter, but the Topology Specific Module™ (TSM™) software should filter the addresses. |
| 11 | Group addressing is supported by a specialized adapter, and TSM checking is not required. |

The `DriverSendRetries` field contains the number of times that the MLID retries send events before aborting the send.

The `DriverLink` field is used by the LSL.

The `DriverSharingFlags` field contains a bit map defining the sharing abilities of the MLID:

0 Set if the LAN board is currently shut down
1 Set if the LAN board can share the primary I/O port
2 Set if the LAN board can share the secondary I/O port
3 Set if the LAN board can share the primary memory range
4 Set if the LAN board can share the secondary memory range
5 Set if the LAN board can share the primary interrupt
6 Set if the LAN board can share the secondary interrupt
7 Set if the LAN board can share DMA channel 0
8 Set if the LAN board can share DMA channel 1
9 Set if there is a command line ifnormation string to place in the AUTOEXEC.NCF file, which is the string that `DriverCommandString` points to
10 This bit is set to prevent default information from being placed into the AUTOEXEC.NCF file when option information is entered on the command line. Setting this bit overrides the setting of bit 9.

The `DriverSlot` field contains the slot number where the LAN board is installed if the board is running in an MCA or EISA machine. Otherwise, this field is 0.

The `DriverIOPortsAndLengths` field contains I/O port information as follows:

| | |
|---|---|
| 1st WORD | The primary base I/O port for the LAN board |
| 2nd WORD | The number of I/O ports beginning with the primary base I/O port |
| 3rd WORD | The secondary base I/O port for the LAN board |
| 4th WORD | The number of I/O ports beginning with the secondary base I/O port |

The `DriverMemoryDecode0` field contains the absolute primary memory address that the LAN board uses (if not used, this field is 0).

The `DriverLength0` field contains the amount of memory (in paragraphs) that the LAN board uses, starting at `DriverMemoryDecode0` (if not used, this field is 0).

The `DriverMemoryDecode1` field contains the absolute secondary memory address that the LAN board uses (if not used, this field is 0).

The `DriverLength1` field contains the amount of memory (in paragraphs) that the LAN board uses, starting at `DriverMemoryDecode0` (if not used, this field is 0).

The first byte in the DriverInterrupt field contains the primary interrupt vector number. The second byte in this field contains the secondary interrupt vector number. FFh = not used.

The first byte in the DriverDMAUsage field contains the primary DMA channel used by the LAN board. The second byte in this field contains the secondary DMA channel used by the LAN board. FFh = not used.

The DriverResourceTag field contains a pointer to the IOResourceTag obtained by the driver.

The DriverConfig field contains a pointer to the LSL copy of the configuration structure (used by the LSL).

The DriverCommandString field contains a long pointer to a new or additional command line string if the driver needs to append something to the command line or replace the default command line in the AUTOEXEC.NCF file. If the driver does not use this option, this field is 0.

The DriverLogicalName field contains the logical name of the LAN driver (given at load time).

The DriverLinearMemory field contains the addresses of DriverMemoryDecode0 and DriverMemoryDecode1. The first LONG contains the linear address of DriverMemoryDecode0. The second LONG contains the linear address of DriverMemoryDecode1.

The DriverChannelNumber field is used for multichannel adapters. It holds the channel number of the NIC to use. The channel number can be specified when a driver is loaded using the "channel=#" keyword (where # is any number greater than 0).

The DriverIOReserved field is reserved for the LSL.

# SSGetLANCustomCounters

Returns custom statistics for a LAN board. For cross-platform functionality, use NWGetLANCustomCountersInfo (page 93) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetLANCustomCounters (
   LONG    boardNumber,
   LONG    startNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**boardNumber**

(IN) Specifies the board number of the LAN board to return counters for.

**startNumber**

(IN) Specifies the custom counter number to start with.

**buffer**

(IN/OUT) Points to a buffer which receives information about LAN custom counters.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

LAN custom counters can keep track of such things as the number of raw sends and fatal retransmissions. Each counter has a string describing the counter and a value associated with the counter.

The `startNumber` parameter is the starting counter number that is returned on this call to SSGetLANCustomCounters. On the first call to this function, `startNumber` should be 0. For subsequent calls, it should be the number of the next counter that has not been retrieved (therefore,

you must keep track of how many counters have been returned). When FALSE is returned in the `moreflag` field of the GetCustomCountersInfoStructure, all custom counters have been retrieved.

This function returns a GetLANConfigInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetCustomCountersInfoStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   moreflag;
   LONG   numberOfCustomCounters;
   BYTE   startOfCustomCounters;
}GetCustomCountersInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `moreflag` field indicates whether there are more custom counters that can be retrieved by further calls to this function. If TRUE is returned, there are more counters; if FALSE is returned, there are no more counters.

The `numberOfCustomCounters` field contains the number of custom counters used by the LAN driver.

The `startOfCustomCounters` field contains the first byte of a number of CustomCountersInfo structures in the buffer.

The CustomCountersInfo structure is defined in NWSERVST.H as follows:

```
typedef struct CustomCountersInfo
{
   LONG   value;
   BYTE   stringLength;
   BYTE   stringStart;
}CustomCountersInfo;
```

The `value` field contains the value of the custom counter.

The `stringLength` field contains the length of the string that starts with `stringStart`. If the `stringLength` is zero, there is no string following.

The `stringStart` field contains the first byte of a string that describes the custom counter.

# SSGetLoadedMediaNumberList

Returns a list of loaded media numbers. (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWGetLoadedMediaNumList (page 95).)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>
#include <nwmediam.h>

LONG  SSGetLoadedMediaNumberList (
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**buffer**

(IN/OUT) Points to a buffer which receives a list of LAN boards.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetMediaNumberListStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetMediaNumberListStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   maxNumberOfMedia;
   LONG   mediaListCount;
   LONG   mediaList;
}GetMediaNumberListStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `maxNumberOfMedia` field contains the maximum number of media allowed.

The `mediaListCount` field contains the number of media in `mediaList`.

The `mediaList` field contains the first media item in the list. More media follow.

# SSGetLSLInfo

Returns information about the LSL. For cross-platform functionality, use NWGetLSLInfo (page 97) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetLSLInfo (
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`buffer`**

(IN/OUT) Points to a buffer which receives a list of LAN boards.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetLSLInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetLSLInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetLSLInfoStructure
{
   LONG             currentServerTime;
   BYTE             vConsoleVersion;
   BYTE             vConsoleRevision;
   WORD             reserved;
   LSLInformation   LSLInfo;
}GetLSLInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The vConsoleVersion field contains the console version number. vConsoleVersion and vConsoleRevision track packet format.

The vConsoleRevision field contains the console version revision number.

The LSLInfo field contians an LSLInformation structure.

The LSLInformation structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   RxBufs;
   LONG   RxBufs75PerCent;
   LONG   RxBufsCheckedOut;
   LONG   RxBufMaxSize;
   LONG   MaxPhysicalSize;
   LONG   LastTimeRxBufAllocated;
   LONG   MaxNumbersOfProtocols;
   LONG   MaxNumbersOfMediaTypes;
   LONG   TotalTXPackets;
   LONG   GetECBBfrs;
   LONG   GetECBFails;
   LONG   AESEventCounts;
   LONG   PostpondedEvents;
   LONG   ECBCxlFails;
   LONG   ValidBfrsReused;
   LONG   EnqueuedSendCnt;
   LONG   TotalRXPackets;
   LONG   UnclaimedPackets;
   BYTE   StatisticsTableMajorVersion;
   BYTE   StatisticsTableMinorVersion;
} LSLInformation;
```

The RxBufs field contains the total number of LSL receive buffers.

The RxBufs75PerCent field contains the number of LSL receive buffers that must be in use before a warning message is issued that buffers are getting low.

The RxBufsCheckedOut field contains the number of LSL buffers in use.

The RxBufMaxSize field contains the size of the data portion of the ECBs in bytes.

The MaxPhysicalSize field contains the total size of the ECB in bytes.

The LastTimeRxBufAllocated field contains the last time (in ticks since the server was brought up) a buffer was checked out.

The MaxNumbersOfProtocols field contains the number of protocol stacks supported by the OS.

The MaxNumbersOfMediaTypes field contains the number of frame types supported by the OS.

The TotalTXPackets field contains the number of packet transmit requests.

The GetECBBfrs field contains the number of ECBs that were requested.

The GetECBFails field contains the number of times an ECB request failed.

The AESEventCounts field contains the total number of AES events that have been processed.

The `PostpondedEvents` field contains the total number of AES events postponed because of critical sections.

The `ECBCxlFails` field contains the number of AES cancel requests that failed because the event was not found on the AES list.

The `ValidBfrsReused` field contains the number of ECBs in the hold queue that were reused before they were removed from the hold queue.

The `EnqueuedSendCnt` field contains the number of send events in the queue that have occurred.

The `TotalRXPackets` field contains the total number of received incoming packets.

The `UnclaimedPackets` field contains the total number of unclaimed incoming packets.

The `StatisticsTableMajorVersion` field contains the major version of the LSL statistics table.

The `StatisticsTableMinorVersion` field contains the minor version of the LSL statistics table.

# SSGetLSLLogicalBoardStats

Returns information about LSL logical boards. For cross-platform functionality, use NWGetLSLLogicalBoardStats (page 99) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetLSLLogicalBoardStats (
   LONG    boardNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**boardNumber**

(IN) Specifies the board number to return information for.

**buffer**

(IN/OUT) Points to a buffer which receives a LSL board information.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be the size of GetLSLBoardStatsStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetLSLInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetLSLBoardStatsStructure
{
   LONG           currentServerTime;
   BYTE           vConsoleVersion;
   BYTE           vConsoleRevision;
   WORD           reserved;
   LogicalBoard   boardStats;
}GetLSLBoardStatsStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `boardStats` field contains an LogicalBoard structure.

The LogicalBoard structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   LogTtlTxPackets;
   LONG   LogTtlRxPackets;
   LONG   LogUnclaimedPackets;
   LONG   reserved;
} LogicalBoard;
```

The `LogTtlTxPackets` field contains the total number of packets transmitted.

The `LogTtlRxPackets` field contains the total number of packets received.

The `LogUnclaimedPackets` field contains the total number of unclaimed packets.

# SSGetMediaManagerObjChildList

Returns a list of children belonging to a given media manager parent object. For cross-platform functionality, use NWGetMediaMgrObjChildrenList (page 101) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetMediaManagerObjChildList (
   LONG    startNumber,
   LONG    objType,
   LONG    parentObjNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startNumber`**

(IN) Specifies the number to start with. This parameter should be 0 on the first call to this function. On subsequent calls, `startNumber` should be the value returned in the `nextStartNumber` field of the GetMMChildListStructure.

**`objType`**

(IN) Specifies the type of object to return information for.

**`parentObjectNumber`**

(IN) Specifies the object number of the media manager object for which you want a list of children.

**`buffer`**

(IN/OUT) Points to a buffer which retruns a GetMMObjectChildListStructure.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

The `objType` parameter specifies the media manager database type:

| | |
|---|---|
| 0 | ADAPTER_OBJECT |
| 1 | CHANGER_OBJECT |
| 2 | RDEVICE_OBJECT |
| 2 | DEVICE_OBJECT |
| 3 | MDEVICE_OBJECT |
| 4 | RMECIA_OBJECT |
| 4 | MEDIA_OBJECT |
| 5 | PARTITION_OBJECT |
| 6 | SLOT_OBJECT |
| 7 | HOTFIX_OBJECT |
| 8 | MIRROR_OBJECT |
| 9 | PARITY_OBJECT |
| 10 | VOLUME_SEG_OBJECT |
| 11 | VOLUME_OBJECT |
| 12 | CLONE_OBJECT |
| 13 | FMEDIA_OBJECT |
| 14 | UNKNOWN_OBJECT |

SSGetMediaManagerObjChildList returns a GetMMObjectChildListStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetMMObjectChildListStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   nextStartNum;
   LONG   objectCount;
   LONG   objects;
}GetMMObjectChildListStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `nextStartNum` field contains the number to pass as the `startNumber` parameter on the next call to this function. When this field is 0, all information has been processed by this function.

The `objectCount` field contains the number of child objects that are in the buffer.

The `objects` field contains the ID number of the first media manager child object. More ID numbers follow.

# SSGetMediaManagerObjInfo

Returns information about media manager objects. For cross-platform functionality, use NWGetMediaMgrObjInfo (page 103) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetMediaManagerObjInfo (
   LONG    objNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**objNumber**

(IN) Specifies the object number of the media manager object that you want information for.

**buffer**

(IN/OUT) Points to a buffer which receives information about the media manager object identified by `objNumber`.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be the size of GetMManagerObjInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetMManagerObjInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetMManagerObjInfoStructure
{
   LONG     currentServerTime;
   BYTE     vConsoleVersion;
   BYTE     vConsoleRevision;
   WORD     reserved;
```

```
   struct   CopyOfGenericInfoDef info;
}GetMManagerObjInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `info` field contains a CopyOfGenericInfoDef structure.

The CopyOfGenericInfoDef structure is defined in NWSERVST.H as follows:

```
struct CopyOfGenericInfoDef
{
   struct CopyOfPMStructure   mediaInfo;
   LONG   mediatype;
   LONG   cartridgetype;
   LONG   unitsize;
   LONG   blocksize;
   LONG   capacity;
   LONG   preferredunitsize;
   BYTE   name[64];
   LONG   type;
   LONG   status;
   LONG   functionmask;
   LONG   controlmask;
   LONG   parentcount;
   LONG   siblingcount;
   LONG   childcount;
   LONG   specificinfosize;
   LONG   objectuniqueid;
   LONG   mediaslot;
};
```

The `mediaInfo` field contains a CopyOfPMStructure.

The `mediatype` field contains the device type of the object:

| | |
|---|---|
| 0 | Hard disk |
| 1 | CD-ROM |
| 2 | WORM device |
| 3 | Tape device |
| 4 | Magneto-Optical (MO) device |

The `cartridgetype` field indicates the type of cartridge or magazine that the device can use. The defined types follow:

| | |
|---|---|
| 0x00000000 | fixed media |

| | |
|---|---|
| 0x00000001 | 5.25 floppy |
| 0x00000002 | 3.5 floppy |
| 0x00000003 | 5.25 optical |
| 0x00000004 | 3.5 optical |
| 0x00000005 | 0.5 tape |
| 0x00000006 | 0.25 tape |
| 0x00000007 | 8 mm tape |
| 0x00000008 | 4 mm tape |
| 0x00000009 | Bernoulli disk |

The `unitsize` field contains the current transfer unit size (in bytes) for the device.

The `blocksize` field contains the size of a block for the device (in bytes).

The `capacity` field contains the capacity of the device (in blocks).

The `preferredunitsize` field contains the preferred transfer unit size (512 bytes to 1K) for the device.

The `name` field contains a length-preceeded string representing the name of the object.

The `type` field contains the media manager database type:

| | |
|---|---|
| 0 | ADAPTER_OBJECT |
| 1 | CHANGER_OBJECT |
| 2 | RDEVICE_OBJECT |
| 2 | DEVICE_OBJECT |
| 3 | MDEVICE_OBJECT |
| 4 | RMECIA_OBJECT |
| 4 | MEDIA_OBJECT |
| 5 | PARTITION_OBJECT |
| 6 | SLOT_OBJECT |
| 7 | HOTFIX_OBJECT |
| 8 | MIRROR_OBJECT |
| 9 | PARITY_OBJECT |
| 10 | VOLUME_SEG_OBJECT |
| 11 | VOLUME_OBJECT |
| 12 | CLONE_OBJECT |
| 13 | FMEDIA_OBJECT |
| 14 | UNKNOWN_OBJECT |

The `status` field contains the status mask for the object:

| | |
|---|---|
| 0x00000001 | OBJECT_ACTIVATED |
| 0x00000002 | OBJECT_PHANTOM |
| 0x00000004 | OBJECT_ASSIGNABLE |
| 0x00000008 | OBJECT_ASSIGNED |
| 0x00000010 | OBJECT_RESERVED |
| 0x00000020 | OBJECT_BEING_IDENTIFIED |
| 0x00010000 | OBJECT_IN_DEVICE |
| 0x00020000 | OBJECT_IN_MAGAZINE |
| 0x00040000 | OBJECT_IN_CHANGER |
| 0x00080000 | OBJECT_LOADABLE |
| 0x00080000 | OBJECT_BEING_LOADED |
| 0x01000000 | OBJECT_DEVICE_LOCK |
| 0x02000000 | OBJECT_CHANGER_LOCK |
| 0x04000000 | OBJECT_REMIRRORING |
| 0x08000000 | OBJECT_SELECTED |

The `functionmask` field indicates the access functions supported on the device:

0 Random read
1 Random write
2 Random write once
3 Sequential read
4 Sequential write
5 Reset end of media
6 Single file marks
7 Multi-file marks
8 Single set marks
9 Multi-set marks
10 Relative data blocks
11 Direct data blocks
12 Position partition
13 Position media

The `controlmask` field indicates the type of control functions that can be issued to the device:

0 Activate/deactivate
1 Mount/dismount
2 Select/unselect
3 Lock/unlock
4 Eject

The `parentcount` field contains the number of parent objects for the device (usually 1).

The `siblingcount` field contains the number of sibling objects for the device (objects with common dependencies).

The `childcount` field contains the number of child objects for the device (objects that depend on the device).

The `specificinfosize` field contains the size of the data structures that are returned.

The `objectuniqueid` field contains the number which identifies the device in the media manager database.

The `mediaslot` field contains the number of the slot that the device occupies.

CopyOfPMStructure is defined in NWSERVST.H as follows:

```
struct CopyOfPMStructure
{
   BYTE   f1[64];
   LONG   f2;
   LONG   f3;
};
```

The `f1` field contains the label of the object.

The `f2` field contains the Novell assigned number for the object.

The `f3` field contains the DOS timestamp of the object.

# SSGetMediaManagerObjList

Returns a list of media manager objects. For cross-platform functionality, use NWGetMediaMgrObjList (page 105) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetMediaManagerObjList (
   LONG    startNumber,
   LONG    objType,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startNumber`**

(IN) Specifies the number to start with. This parameter should be -1 for the first call to this function. On subsequent calls, `startNumber` should be the value returned in the `nextStartNum` field of the GetMMObjectListsStructure.

**`objType`**

(IN) Specifies the type of object to return information for.

**`buffer`**

(IN/OUT) Points to a buffer which receives a list of media manager objects.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

The `objType` parameter specifies the media manager database type:

| | |
|---|---|
| 0 | ADAPTER_OBJECT |

| | |
|---|---|
| 1 | CHANGER_OBJECT |
| 2 | RDEVICE_OBJECT |
| 2 | DEVICE_OBJECT |
| 3 | MDEVICE_OBJECT |
| 4 | RMECIA_OBJECT |
| 4 | MEDIA_OBJECT |
| 5 | PARTITION_OBJECT |
| 6 | SLOT_OBJECT |
| 7 | HOTFIX_OBJECT |
| 8 | MIRROR_OBJECT |
| 9 | PARITY_OBJECT |
| 10 | VOLUME_SEG_OBJECT |
| 11 | VOLUME_OBJECT |
| 12 | CLONE_OBJECT |
| 13 | FMEDIA_OBJECT |
| 14 | UNKNOWN_OBJECT |

This function returns a GetMMObjectListsStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetMMObjectListsStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   nextStartNum;
   LONG   objectCount;
   LONG   objects;
}GetMMObjectListsStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `nextStartNum` field contains the number to be passed as the `startNumber` parameter on the next call to this function. When this field is -1, all information has been processed.

The `objectCount` field contains the number of media manager objects in the buffer.

The `objects` field contains the ID number of the first media manager object. More ID numbers follow.

# SSGetMediaNameByNumber

Returns a media name for a given media number. For cross-platform functionality, use NWGetMediaNameByMediaNum (page 108) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetMediaNameByNumber (
   LONG    mediaNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**mediaNumber**

(IN) Specifies the ID number of the media for which you want a name.

**buffer**

(IN/OUT) Points to a buffer which receives the media name.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetMediaNameByNumberStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetMediaNameByNumberStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   BYTE   mediaNameLength;
```

```
   BYTE   mediaName;
}GetMediaNameByNumberStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `mediaNameLength` field contains the length of the media name.

The `mediaName` structure contains the first byte of the media name.

# SSGetNetRouterInfo

Returns information about network routing on a server. For cross-platform functionality, use NWGetNetworkRouterInfo (page 116) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetNetRouterInfo (
   LONG    networkNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**networkNumber**

(IN) Specifies the network to return information for.

**buffer**

(IN/OUT) Points to a buffer which receives routing information for the server.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be the size of GetNetRouterInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetNetRouterInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetNetRouterInfoStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   netIDNumber;
   WORD   hopsToNet;
```

```
   WORD   netStatus;
   WORD   timeToNet;
}GetNetRouterInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `netIDNumber` field contains the network ID number.

The `hopsToNet` field contains the number of hops to the network.

The `netStatus` field indicates the status of the network.

The `timeToNet` field contains the number of ticks to the network (round-trip).

# SSGetNetworkRoutersInfo

Returns information about the routers on a network. For cross-platform functionality, use NWGetNetworkRoutersInfo (page 118) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetNetworkRoutersInfo (
   LONG    networkNumber,
   LONG    startNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**networkNumber**

(IN) Specifies the network number to return router information for.

**startNumber**

(IN) Specifies the number to start with.

**buffer**

(IN/OUT) Points to a buffer which receives a information about routers on a network.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetNetworkRoutersInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetNetworkRoutersInfoStructure
{
   LONG                    currentServerTime;
   BYTE                    vConsoleVersion;
```

```
   BYTE                    vConsoleRevision;
   WORD                    reserved;
   LONG                    numberOfEntries;
   RoutersInfoStructure    info;
}GetNetworkRoutersInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `numberOfEntries` field contains the number of RoutersInfoStructure structures in the buffer.

The `info` field contains the first RoutersInfoStructure. More follow.

The RoutersInfoStructure is defined in NWSERVST.H as follows:

```
typedef struct RoutersInfoStructure
{
   BYTE   node[6];
   LONG   connectedLAN;
   WORD   hopsToNetCount;
   WORD   timeToNet;
}RoutersInfoStructure;
```

The `node` field contains the 6-byte network address of the router.

The `connectedLAN` field contains the LAN board number of the router.

The `hopsToNetCount` field contains the number of hops to the network specified by `networkNumber`.

The `timeToNet` field contains the time (in ticks) to the network specified by `networkNumber`.

# SSGetNLMInfo

Returns information about an NLM running on a server. For cross-platform functionality, use NWGetNLMInfo (page 122) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetNLMInfo (
   LONG    NLMNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`NLMNumber`**

(IN) Specifies the number assigned to the NLM by the OS when the NLM was loaded.

**`buffer`**

(IN/OUT) Points to a buffer which receives NLM information.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns information for the current connection. The `NLMNumber` parameter is the number assigned to that NLM by the OS when it is loaded. This number can be obtained by calling SSGetNLMLoadedList.

This function returns a GetNLMInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetNLMInfoStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
```

```
   BYTE   vConsoleRevision;
   WORD   reserved;
   NLMInformation   NLMInfo;
   BYTE   startOfLStrings;
}GetNLMInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `NLMInfo` field contains an NLMInformation structure.

The `startOfLStrings` field contains the first byte of three length-preceded strings (the strings can be 0 bytes long). The strings represent the filename, name, and copyright of the NLM.

The NLMInformation structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   nlmIdentificationNumber;
   LONG   nlmFlags;
   LONG   nlmType;
   LONG   nlmParentID;
   LONG   nlmMajorVersion;
   LONG   nlmMinorVersion;
   LONG   nlmRevision;
   LONG   nlmYear;
   LONG   nlmMonth;
   LONG   nlmDay;
   LONG   nlmAllocAvailBytes;
   LONG   nlmAllocFreeCount;
   LONG   nlmLastGarbCollect;
   LONG   nlmMessageLanguage;
   LONG   nlmNumberOfReferencedPublics;
} NLMInformation;
```

The `nlmIdentificationNumber` field contains the number assigned to the NLM when it was loaded.

The `nlmFlags` field contains a bit mask with a combination of the following:

| | |
|---|---|
| 0x0001 | REENTRANT: The module is reentrant. That is, if the NLM is loaded twice, the actual code in the server's memory is reused. |
| 0x0002 | MULTIPLE: The module can be loaded more than once by a LOAD console command. |
| 0x0004 | SYNCHRONIZE: This NLM option causes the load process to go to sleep until the NLM calls SynchronizeStart.This prevents any other console commands from being processed (particularly LOAD commands) while the NLM is being loaded. |
| 0x0008 | PSEUDOPREEMPTION: This option requests that the OS force the NLM to relinquish control if the NLM does not do so on its own often enough. |

The `nlmType` field contains a number indicating the NLM type:

| Number | Extension | Description |
|---|---|---|
| 0 | .NLM | Generic NLM (default value) |
| 1 | .LAN | LAN driver |
| 2 | .DSK | Disk driver |
| 3 | .NAM | Name space support module |
| 4 | .NLM | Utility or support program |
| 5 | .MSL | Mirrored Server Link™ (MSL™) |
| 6 | .NLM | OS NLM |
| 7 | .NLM | Paged high OS NLM |
| 8 | .HAM | Host Adapter Module (works with Custom Device Module) |
| 9 | .CDM | Custom Device Module (works with Host Adapter Module) |
| 10 | .NLM | OS Reserved |
| 11 | .NLM | OS Reserved |
| 12 | .NLM | OS Reserved |
| 13 | | SMP normal |
| 14 | | NIOS |
| 15 | .CAD | CIOS CAD |
| 16 | .CLS | CIOS CLS |
| 20 through 32 | | Reserved for NICI (Novell International Cryptographic Infrastructure) |

The `nlmParentID` field contains the number assigned to the NLM that caused this NLM to be loaded.

The `nlmMajorVersion` field contains the major version number of the NLM.

The `nlmMinorVersion` field contains the minor version number of the NLM.

The `nlmRevision` field contains the revision letter of the NLM.

The `nlmYear`, `nlmMonth`, and `nlmDay` fields contain the year, month, and day that the NLM was created on.

The `nlmAllocAvailBytes`, `nlmAllocFreeCount`, and `nlmLastGarbCollect` field contain garbage collection information. The `nlmAllocAvailBytes` contains the bytes available for allocation by the NLM. The `nlmAllocFreeCount` contain the number of bytes freed that can be reclaimed. The `nlmLastGarbCollect` field contains the last time garbage collection was done for the NLM (expressed in ticks since the server was brought up).

The `nlmMessageLanguage` field contains the number representing the language that the NLM uses.

The `nlmNumberOfReferencedPublics` field contains the number of external symbols referenced by the NLM.

# SSGetNLMLoadedList

Returns a list of NLM applications running on a server. For cross-platform functionality, use NWGetNLMLoadedList (page 124) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetNLMLoadedList (
   LONG    startNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startNumber`**

(IN) Specifies the NLM number to start with.

**`buffer`**

(IN/OUT) Points to a buffer which receives NLM information.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a list of NLM applications for the current connection. The list is returned in a GetNLMLoadedListStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetNLMLoadedListStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   NLMLoadedCount;
   LONG   NLMCount;
```

```
   LONG    NLMNumbers;
}GetNLMLoadedListStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `NLMLoadedCount` field contains the number of NLM applications loaded on the server. This number might indicate NLM applications that you cannot get information for, such as the loader. Therefore, the total number of NLM applications you receive information about after iterative calls to this function might be less than the `NLMLoadedCount`.

The `NLMCount` field contains the number of NLM applications listed in the buffer.

The `NLMNumbers` field contains the numbers assigned to NLM applications loaded on the server.

# SSGetNLMResourceTagList

Returns information about resources used by NLM applications on a server. For cross-platform functionality, use NWGetNLMsResourceTagList (page 126) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetNLMResourceTagList (
   LONG    NLMNumber,
   LONG    startNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**NLMNumber**

(IN) Specifies the number assigned to the NLM by the OS when the NLM was loaded

**startNumber**

(IN) Specifies the resource tag to start with.

**buffer**

(IN/OUT) Points to a buffer which receives a resource tag information.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetNLMResourceTagList in `buffer`. This structure is defined in NWSERVST.H as follows:

```
ttypedef struct GetNLMResourceTagList
{
   LONG            currentServerTime;
   BYTE            vConsoleVersion;
```

```
   BYTE            vConsoleRevision;
   WORD            reserved;
   LONG            totalNumOfRTags;
   LONG            currentNumOfRTags;
   RTagStructure   RTagStart;
}GetNLMResourceTagList;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `totalNumOfRTags` field contains the total number of resource tags that the NLM is using.

The `currentNumOfRTags` field contains the number of resource tags that the buffer contains. If additional tags are used by the NLM that did not fit in the buffer, call this function again, using the next number as the `startNumber`.

The `RTagStart` field contains an RTagStructure. This is the first RTagStructure in the buffer. More structures follow, one for each resource tag used on the server.

RTagStructure is defined in NWSERVST.H as follows:

```
typedef struct RTagStructure
{
   LONG   rTagNumber;
   LONG   signature;
   LONG   count;
   BYTE   name;
}RTagStructure;
```

The `rTagNumber` field contains the number assigned to the resource tag when it was allocated.

The `signature` field contains the signature of the resource tag.

The `count` field contains the number of this kind of tag that has been allocated.

The `name` field contains the name of the resource tag. It is null terminated string of unknown size.

# 11.5 SSGetO*-SSGetV* Functions

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

# SSGetOSVersionInfo

Returns information about the NetWare OS version running on the server. For cross-platform functionality, use NWGetOSVersionInfo (page 128) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetOSVersionInfo (
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**buffer**

(IN/OUT) Points to a buffer which receives OS version information.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be the size of GetOSVersionInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetOSVersionInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetOSVersionInfoStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   BYTE   OSMajorVersion;
   BYTE   OSMinorVersion;
   BYTE   OSRevision;
   BYTE   accountVersion;
   BYTE   VAPVersion;
   BYTE   queueingVersion;
```

```
   BYTE   securityRestLvl;
   BYTE   bridgingSupport;
   LONG   maxNumOfVol;
   LONG   maxNumOfConn;
   LONG   maxNumOfUsers;
   LONG   maxNumOfnameSpaces;
   LONG   maxNumOfLANS;
   LONG   maxNumOfMedias;
   LONG   maxNumOfStacks;
   LONG   maxDirDepth;
   LONG   maxDataStreams;
   LONG   maxNumOfSpoolPr;
   LONG   serverSerialNumber;
   WORD   serverApplicationNumber;
}GetOSVersionInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `OSMajorVersion` field contains the major version number of the OS.

The `OSMinorVersion` field contains the minor version number of the OS.

The `OSRevision` field contains the version revision letter of the OS.

The `accountVersion` field contains the version of the accounting subsystem.

The `VAPVersion` field is not used.

The `queueingVersion` field contains the Queuing version number.

The `securityRestLvl` field contains the Security Restriction version number.

The `bridgingSupport` field contains the Internet Bridge support version number.

The `maxNumOfVol` field contains the maximum number of volumes that can be simultaneously mounted on the server. Versions 4.10 and higher return 255 in this field. 0-63 are physical volumes and 64-255 are logical volumes.

The `maxNumOfConn` field contains the maximum number of connections that can be used simultaneously on the server.

The `maxNumOfUsers` field contains the maximum number of simultaneous users allowed on the server.

The `maxNumOfnameSpaces` field contains the maximum number of name spaces that can be simultaneously loaded on a server.

The `maxNumOfLANS` field contains the maximum number of LAN boards that can be used on the server.

The `maxNumOfMedias` field contains the maximum number of different media types allowed on the server.

The `maxNumOfStacks` field contains the maximum number of protocol stacks that can be used in the server.

The `maxDirDepth` field contains the maximum depth of directories that can be used on the server.

The `maxDataStreams` field contains the maximum number of data streams that can be used on the server.

The `maxNumOfSpoolPr` field contains the maximum number of spool printers that can be used on the server.

The `serverSerialNumber` field contains the serial number of the server.

The `serverApplicationNumber` field is included for backward compatibility.

# SSGetPacketBurstInfo

Returns Packet Burst™ information. For cross-platform functionality, use NWGetPacketBurstInfo (page 130) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetPacketBurstInfo (
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`buffer`**

(IN/OUT) Points to a buffer which receives Packet Burst information.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetPacketBurstInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetPacketBurstInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetPacketBurstInfoStructure
{
   LONG                    currentServerTime;
   BYTE                    vConsoleVersion;
   BYTE                    vConsoleRevision;
   WORD                    reserved;
   PacketBurstInformation  packetBurstInfo;
}GetPacketBurstInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The vConsoleVersion field contains the console version number. vConsoleVersion and vConsoleRevision track packet format.

The vConsoleRevision field contains the console version revision number.

The packetBurstInfo field contains a PacketBurstInformation structure. This structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   BigInvalidSlotCount;
   LONG   BigForgedPacketCount;
   LONG   BigInvalidPacketCount;
   LONG   BigStillTransmittingCount;
   LONG   StillDoingTheLastRequestCount;
   LONG   InvalidControlRequestCount;
   LONG   ControlInvalidMessageNumberCount;
   LONG   ControlBeingTornDownCount;
   LONG   BigRepeatTheFileReadCount;
   LONG   BigSendExtraCCCount;
   LONG   BigReturnAbortMessageCount;
   LONG   BigReadInvalidMessageNumberCount;
   LONG   BigReadDoItOverCount;
   LONG   BigReadBeingTornDownCount;
   LONG   PreviousControlPacketCount;
   LONG   SendHoldOffMessageCount;
   LONG   BigReadNoDataAvailableCount;
   LONG   BigReadTryingToReadTooMuchCount;
   LONG   ASyncReadErrorCount;
   LONG   BigReadPhysicalReadErrorCount;
   LONG   ControlBadACKFragmentListCount;
   LONG   ControlNoDataReadCount;
   LONG   WriteDuplicateRequestCount;
   LONG   ShouldntBeACKingHereCount;
   LONG   WriteInconsistentPacketLengthsCount;
   LONG   FirstPacketIsntAWriteCount;
   LONG   WriteTrashedDuplicateRequestCount;
   LONG   BigWriteInvalidMessageNumberCount;
   LONG   BigWriteBeingTornDownCount;
   LONG   BigWriteBeingAbortedCount;
   LONG   ZeroACKFragmentCountCount;
   LONG   WriteCurrentlyTransmittingCount;
   LONG   TryingToWriteTooMuchCount;
   LONG   WriteOutOfMemoryForControlNodesCount;
   LONG   WriteDidntNeedThisFragmentCount;
   LONG   WriteTooManyBuffersCheckedOutCount;
   LONG   WriteTimeOutCount;
   LONG   WriteGotAnACKCount;
   LONG   WriteGotAnACKCount1;
   LONG   PollerAbortedTheConnectionCount;
   LONG   MaybeHadOutOfOrderWritesCount;
   LONG   HadAnOutOfOrderWriteCount;
   LONG   MovedTheACKBitDownCount;
   LONG   BumpedOutOfOrderWriteCount;
   LONG   PollerRemovedOldOutOfOrderCount;
```

```
   LONG   WriteDidntNeedButRequestedACKCount;
   LONG   WriteTrashedPacketCount;
   LONG   TooManyACKFragmentsCount;
   LONG   SavedAnOutOfOrderPacketCount;
   LONG   ConnectionBeingAbortedCount;
} PacketBurstInformation;
```

Descriptions of the fields in this structure are not available as of this edition.

# SSGetProtocolConfiguration

Returns configuration information about the protocols on a server. For cross-platform functionality, use NWGetProtocolStackConfigInfo (page 133) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetProtocolConfiguration (
   LONG    startNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startNumber`**

(IN) Indicates the number to start with.

**`buffer`**

(IN/OUT) Points to a buffer which receives a information about protocol configuration of the server.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetProtocolConfigStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetProtocolConfigStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   BYTE   configMajorVersion;
```

```
   BYTE   configMinorVerstion;
   BYTE   stackMajorVersion;
   BYTE   stackMinorVersion;
   BYTE   shortName[16];
   BYTE   fullNameLength;
   BYTE   fullName;
}GetProtocolConfigStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `configMajorVersion` field contains the major version number of the configuration table.

The `configMinorVerstion` field contains the minor version number of the configuration table.

The `stackMajorVersion` field contains the major version number of the protocol stack.

The `stackMinorVersion` field contains the minor version number of the protocol stack.

The `shortName` field contains the short protocol name (used to register the stack with the LSL):

| | |
|---|---|
| VIRTUAL_LAN | For use where no Frame ID/MAC envelope is necessary |
| LOCALTALK | Apple LocalTalk frame |
| ETHERNET_II | Ethernet using a DEC Ethernet II envelope |
| ETHERNET_802.2 | Ethernet (802.3) using an 802.2 envelope |
| TOKEN-RING | Token ring (802.5) using an 802.2 envelope |
| ETHERNET_802.3 | IPX 802.3 raw encapsulation |
| 802.4 | Token-passing bus envelope |
| NOVELL_PCN2 | Novell's IBM PC Network II envelope |
| GNET | Gateway's GNET frame envelope |
| PRONET-10 | Proteon's ProNET I/O frame envelope |
| ETHERNET_SNAP | Ethernet (802.3) using an 802.2 envelope with SNAP |
| TOKEN-RING_SNAP | Token ring (802.5) using an 802.2 envelope with SNAP |
| LANPAC_II | Racore's frame envelope |
| ISDN | Integrated Services Digital Network |
| NOVELL_RX-NET | Novell's ArcNet 68 envelope |
| IBM_PCN2_802.2 | IBM PCN2 using 802.2 envelope |
| IBM_PCN2_SNAP | IBM PCN2 using 802.2 with SNAP envelope |
| OMNINET/4 | Corvus's frame envelope |

| | |
|---|---|
| 3270_COAXA | Harris Adacom's frame envelope |
| IP | IP Tunnel frame envelope |
| FDDI_802.2 | FDDI (802.7) using an 802.2 envelope |
| IVDLAN_802.9 | Commtex, Inc.'s frame envelope |
| DATACO_OSI | Dataco's frame envelope |
| FDDI_SNAP | FDDI (802.7) using 802.2 with a SNAP envelope |
| IBM_SDLC | SDLC tunnel envelope |
| PCO_FDDITP | PC Office frame envelope |
| WAIDNET | Hypercommunications |
| SLIP | Novell frame envelope |
| PPP | Novell frame envelope |

The `fullNameLength` field contains the length of the full name.

The `fullName` field contains the first byte of the full name.

# SSGetProtocolCustomInfo

Returns custom information about a protocol stack on a server. For cross-platform functionality, use NWGetProtocolStackCustomInfo (page 135) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetProtocolCustomInfo (
   LONG    stackNumber,
   LONG    customStartNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**stackNumber**

(IN) Specifies the number of the protocol stack that you want information for.

**customStartNumber**

(IN) Specifies the custom information structure to start with. For the first call to this function, `customStartNumber` should be 0. On subsequent calls, it should be the next custom information structure that has not been retrieved. All information has been retrieved when no more information is returned by this function.

**buffer**

(IN/OUT) Points to a buffer which receives custom information about the protocol stack.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetProtocolCustomInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetProtocolCustomInfoStructure
{
   LONG                currentServerTime;
   BYTE                vConsoleVersion;
   BYTE                vConsoleRevision;
   WORD                reserved;
   LONG                customCount;
   ProtocolCustomInfo  info;
}GetProtocolCustomInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `customCount` field contains the number of ProtocolCustomInfo structures in the buffer.

The `info` field contains the first ProtocolCustomInfo structure. More structures follow.

The ProtocolCustomInfo structure is defined in NWSERVST.H as follows:

```
typedef struct ProtocolCustomInfo
{
   LONG   value;
   BYTE   length;
   BYTE   customData;
}ProtocolCustomInfo;
```

The `value` field contains the value of the custom counter.

The `length` field contains the length of `customData`.

The `customData` field contains the first byte of a string describing the custom counter. The length of the string is `length`.

# SSGetProtocolNumbersByLANBoard

Returns a list of protocol stack ID numbers for a given LAN board. For cross-platform functionality, use NWGetProtocolStkNumsByLANBrdNum (page 139) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetProtocolNumbersByLANBoard (
   LONG    LANBoardNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**LANBoardNumber**

(IN) Specifies the ID number of the LAN board for which you want a list of protocols.

**buffer**

(IN/OUT) Points to a buffer which receives a list of protocols.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetProtocolByBoardStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetProtocolByBoardStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   stackIDCount;
```

```
   LONG   stackID;
}GetProtocolByBoardStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `stackIDCount` field contains the number of protocol stack ID numbers in the buffer.

The `stackID` field contains the first protocol stack ID number in the buffer. More numbers follow.

# SSGetProtocolNumbersByMedia

Returns a list of protocol stack ID numbers for a given media. For cross-platform functionality, use NWGetProtocolStkNumsByMediaNum (page 141) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetProtocolNumbersByMedia (
   LONG    mediaNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**mediaNumber**

(IN) Specifies the media number (frame number) for which you want protocol stack information.

**buffer**

(IN/OUT) Points to a buffer which receives a list of protocols.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a list of protocol stack ID numbers for a given media. The `mediaNumber` is the frame ID number of the media:

| | |
|---|---|
| 0 | For use where no Frame ID/MAC envelope is necessary |
| 1 | Apple LocalTalk frame |
| 2 | Ethernet using a DEC Ethernet II envelope |
| 3 | Ethernet (802.3) using an 802.2 envelope |

| | |
|---|---|
| 4 | Token ring (802.5) using an 802.2 envelope |
| 5 | IPX 802.3 raw encapsulation |
| 6 | Token-passing bus envelope |
| 7 | Novell's IBM PC Network II envelope |
| 8 | Gateway's GNET frame envelope |
| 9 | Proteon's ProNET I/O frame envelope |
| 10 | Ethernet (802.3) using an 802.2 envelope with SNAP |
| 11 | Token ring (802.5) using an 802.2 envelope with SNAP |
| 12 | Racore's frame envelope |
| 13 | Integrated Services Digital Network |
| 14 | Novell's ArcNet 68 envelope |
| 15 | IBM PCN2 using 802.2 envelope |
| 16 | IBM PCN2 using 802.2 with SNAP envelope |
| 17 | Corvus's frame envelope |
| 18 | Harris Adacom's frame envelope |
| 19 | IP Tunnel frame envelope |
| 20 | FDDI (802.7) using an 802.2 envelope |
| 21 | Commtex, Inc.'s frame envelope |
| 22 | Dataco's frame envelope |
| 23 | FDDI (802.7) using 802.2 with a SNAP envelope |
| 24 | SDLC tunnel envelope |
| 25 | PC Office frame envelope |
| 26 | Hypercommunications |
| 27 | Novell frame envelope |

This function returns a GetProtocolByMediaStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetProtocolByMediaStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   stackIDCount;
   LONG   stackID;
}GetProtocolByMediaStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `stackIDCount` field contains the number of protocol stack ID numbers in the buffer.

The `stackID` field contains the first protocol stack ID number in the buffer. More numbers follow.

# SSGetProtocolStatistics

Returns protocol statistics for a server. For cross-platform functionality, use NWGetProtocolStackStatsInfo (page 137) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetProtocolStatistics (
   LONG    stackNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`stackNumber`**

(IN) Specifies the stack number of the protocol stack for which you want information.

**`buffer`**

(IN/OUT) Points to a buffer which receives protocol statistics.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetProtocolStatsStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetProtocolStatsStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetProtocolStatsStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   BYTE   StatMajorVersion;
   BYTE   StatMinorVersion;
```

```
    WORD   GenericCounters;
    LONG   ValidCntsMask;
    LONG   TotalTxPackets;
    LONG   TotalRxPackets;
    LONG   IgnoredRxPackets;
    WORD   NumberOfCustomCounters;
}GetProtocolStatsStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `StatMajorVersion` field contains the major version number of the statistics table.

The `StatMinorVersion` field contains the minor version number of the statistics table.

The `GenericCounters` field contains the number of counters in the fixed portion of the table. It is always set to 3.

The `ValidCntsMask` field contains a bit mask indicating which counters are valid (starting with right-most representing the first counter). If the bit is 0, the counter is valid; if it is 1, the counter is not valid.

The `TotalTxPackets` field contains the total number of packets that were requested to be transmitted.

The `TotalRxPackets` field contains the total number of packets that were received.

The `IgnoredRxPackets` field contains the number of incoming packets that were ignored by the stack.

The `NumberOfCustomCounters` field contains the number of custom counters for the protocol stack.

# SSGetRouterAndSAPInfo

Returns router and SAP information. For cross-platform functionality, use NWGetGeneralRouterAndSAPInfo (page 81) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetRouterAndSAPInfo (
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**buffer**

(IN/OUT) Points to a buffer which receives router and SAP information for a server.

**bufferLen**

(IN) Specifies the size of `buffer`. This should be the size of GetRouterAndSAPInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetRouterAndSAPInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetRouterAndSAPInfoStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   RIPSocketNumber;
   LONG   routerDownFlag;
   LONG   trackOnFlag;
   LONG   extRouterActiveFlag;
   LONG   SAPSocketNumber;
```

```
   LONG   rpyNearestServerFlag;
}GetRouterAndSAPInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `RIPSocketNumber` field contains the router socket number.

The `routerDownFlag` field indicates whether the internal router is up or down.

The `trackOnFlag` field indicates whether router tracking is active (the console operator issued the TRACK ON console command).

The `extRouterActiveFlag` field indicates whether an external router is active.

The `SAPSocketNumber` field contains the number of the socket that receives SAP packets.

The `rpyNearestServerFlag` field indicates whether the server responds to GetNearestServer.

# SSGetServerInfo

Returns information about a server. For cross-platform functionality, use NWGetServerInfo (page 147) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetServerInfo (
   LONG    serverType,
   BYTE    nameLength,
   BYTE   *name,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`serverType`**

(IN) Specifies the server type. This *must* be 0x400.

**`nameLength`**

(IN) Specifies the length of the server name ( `name`).

**`name`**

(IN) Points to the name of the server.

**`buffer`**

(IN/OUT) Points to a buffer which receives a information about the server.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be the size of GetServerInfoStructure.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetServerInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetServerInfoStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   BYTE   serverAddress[12];
   WORD   hopsToServer;
}GetServerInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `serverAddress` field contains the node address of the server.

The `hopsToServer` field contains the number of hops to the server.

The value of the `serverType` parameter must be 0x0400.

# SSGetServerSourcesInfo

Returns address information about servers known to a server with a given name. For cross-platform functionality, use NWGetServerSourcesInfo (page 153) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetServerSourcesInfo (
   LONG    startNumber,
   LONG    serverType,
   BYTE    nameLength,
   BYTE   *name,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startNumber`**

(IN) Indicates the number to start with.

**`serverType`**

(IN) Specifies the type of server to obtain information for.

**`nameLength`**

(IN) Specifies the length of the server name ( `name`).

**`name`**

(IN) Points to the name of the server.

**`buffer`**

(IN/OUT) Points to a buffer which receives a information about protocol configuration of the server.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetServerSourcesStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetServerSourcesStructure
{
   LONG                        currentServerTime;
   BYTE                        vConsoleVersion;
   BYTE                        vConsoleRevision;
   WORD                        reserved;
   LONG                        numberOfEntries;
   ServerSourceInfoStructure   info;
}GetServerSourcesStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `numberOfEntries` field contains the number of structures in the buffer.

The `info` field contains the first ServerSourceInfoStructure. More follow.

```
typedef struct ServerSourceInfoStructure
{
   BYTE   serverNode[6];
   LONG   connectLAN;
   WORD   hopCount;
}ServerSourceInfoStructure;
```

The `serverNode` field contains the node address of a server.

The `connectLAN` field contains the LAN board number of the server.

The `hopCount` field contains the number of hops to the server.

# SSGetUserInfo

Returns user information for a given connection. For cross-platform functionality, use NWGetUserInfo (page 155) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetUserInfo (
   LONG    connectionNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`connectionNumber`**

(IN) Specifies the connection number that you want user information for. The user for the connection must be logged in (licensed) for the server to return any information about the user.

**`buffer`**

(IN/OUT) Points to a buffer which receives user information.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetUserInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetUserInfoStructure
{
   LONG              currentServerTime;
   BYTE              vConsoleVersion;
   BYTE              vConsoleRevision;
```

```
   WORD              reserved;
   UserInformation   userInfo;
   BYTE              userNameLen;
   BYTE              username;
}GetUserInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `userInfo` field contains a UserInformation structure for the connection.

The `userNameLen` field contains the length of the `username` field.

The `username` field contains the name of the user.

The UserInformation structure is defined in NWSERVST.H as follows:

```
typedef struct {
   LONG   connectionNumber;
   LONG   useCount;
   BYTE   connectionServiceType;
   BYTE   loginTime[7];
   LONG   status;
   LONG   expirationTime;
   LONG   objectType;
   BYTE   transactionFlag;
   BYTE   logicalLockThreshold;
   BYTE   recordLockThreshold;
   BYTE   fileWriteFlags;
   BYTE   fileWriteState;
   BYTE   filler;
   WORD   fileLockCount;
   WORD   recordLockCount;
   BYTE   totalBytesRead[6];
   BYTE   totalBytesWritten[6];
   LONG   totalRequests;
   LONG   heldRequests;
   BYTE   heldBytesRead[6];
   BYTE   heldBytesWritten[6];
} UserInformation;
```

The `connectionNumber` field contains the connection number of the user.

The `useCount` field contains 1 if the connection is in use, or 0 if the connection is not in use.

The `connectionServiceType` field contains the connection type:

| | |
|---|---|
| 1 | (Included for CLIB backwards compatibility) |
| 2 | NCP_CONNECTION_TYPE |

| | |
|---|---|
| 3 | NLM_CONNECTION_TYPE |
| 4 | AFP_CONNECTION_TYPE |
| 5 | FTAM_CONNECTION_TYPE |
| 6 | ANCP_CONNECTION_TYPE |

The `loginTime` field contains the time that the user logged in.

The `status` field contains the status of the connection:

| | |
|---|---|
| 0x00000001 | LOGGED_IN |
| 0x00000002 | BEING_ABORTED |
| 0x00000004 | AUDITED |
| 0x00000008 | NEEDS_SECURITY_CHANGE |
| 0x00000010 | MAC_STATION |
| 0x00000020 | AUTHENTICATED_TEMPORARY |
| 0x00000040 | AUDIT_CONNECTION_RECORDED |
| 0x00000080 | DSAUDIT_CONNECTION_RECORDED |

The `expirationTime` field contains the time before the authentication expires (in ticks). If this value is 0, there is no expiration time.

The `objectType` field contains the type of the user.

The `transactionFlag` field contains transaction tracking information

The `logicalLockThreshold` field contains the maximum number of logical locks the user can have.

The `recordLockThreshold` field contains maximum number of record locks the user can have.

The `fileWriteFlags` field contains a flag indicating the writing status as follows:

| | |
|---|---|
| 1 | Writing |
| 2 | Write aborted |

The `fileWriteState` field indicates the writing status:

| | |
|---|---|
| 0 | not writing |
| 1 | write in progress |
| 2 | write being stopped |

The `filler` field is unused.

The `fileLockCount` field contains the number of files the user has locked.

The `recordLockCount` field contains the number of records the user has locked.

The `totalBytesRead` field contains the number of bytes the user has read (48-bit value).

The `totalBytesWritten` field contains the number of bytes the user has written (48-bit value).

The `totalRequests` field contains the number of requests the user has sent.

The `heldRequests` field contains the number of requests held for accounting purposes.

The `heldBytesRead` field contains the number of bytes the user has read that have a hold on them for accounting purposes.

The `heldBytesWritten` field contains the number of bytes the user has written that have a hold on them

# SSGetVolumeSegmentList

Returns a list of volume segments for a given volume. For cross-platform functionality, use NWGetVolumeSegmentList (page 159) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetVolumeSegmentList (
   LONG    volumeNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`volumeNumber`**

(IN) Specifies the volume number of the volume that you want information for.

**`buffer`**

(IN/OUT) Points to a buffer which receives a list of volume segments.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetVolumeSegmentListStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetVolumeSegmentListStructure
{
   LONG                    currentServerTime;
   BYTE                    vConsoleVersion;
   BYTE                    vConsoleRevision;
   WORD                    reserved;
   LONG                    numberOfSegments;
```

```
   VolumeSegmentStructure   segment;
}GetVolumeSegmentListStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `numberOfSegments` field contains the number of volume segments on the volume.

The `segment` field contains the VolumeSegmentStructure for the first volume segment. More of these structures follow in the buffer, one for each volume segment.

The VolumeSegmentStructure is defined in NWSERVST.H as follows:

```
typedef struct VolumeSegmentStructure
{
   LONG   segmentDevice;
   LONG   segmentOffset;
   LONG   segmentSize;
}VolumeSegmentStructure;
```

The `segmentDevice` field identifies the device that the segment is located on.

The `segmentOffset` field contains the offset of the segment in bytes.

The `segmentSize` field contains the segment size in bytes.

# SSGetVolumeSwitchInfo

Returns information about the number of times various code paths have been taken in the NetWare OS. For cross-platform functionality, use NWGetVolumeSwitchInfo (page 161) instead.

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based Server Environment

## Syntax

```
#include <nit\nwservst.h>

LONG SSGetVolumeSwitchInfo (
   LONG    startNumber,
   BYTE   *buffer,
   LONG    bufferLen);
```

## Parameters

**`startNumber`**

(IN) Specifies the number of the counter to start with. (This parameter is included for the future, when there might be more counters than can be retrieved by one call to this function.)

**`buffer`**

(IN/OUT) Points to a buffer which receives volume switch information.

**`bufferLen`**

(IN) Specifies the size of `buffer`. This should be SS_DEFAULT_BUFFER_SIZE.

## Return Values

ESUCCESS or NetWare errors.

## Remarks

This function returns a GetVolumeSwitchInfoStructure in `buffer`. This structure is defined in NWSERVST.H as follows:

```
typedef struct GetVolumeSwitchInfoStructure
{
   LONG   currentServerTime;
   BYTE   vConsoleVersion;
   BYTE   vConsoleRevision;
   WORD   reserved;
   LONG   totalLFSCounters;
```

```
   LONG   currentLFSCounters;
   LONG   counters;
}GetVolumeSwitchInfoStructure;
```

The `currentServerTime` field contains the time elapsed since the server was brought up. This time is returned in ticks (approximately 1/18 of a second). When this field reaches 0xFFFFFFFF, it wraps to zero.

The `vConsoleVersion` field contains the console version number. `vConsoleVersion` and `vConsoleRevision` track packet format.

The `vConsoleRevision` field contains the console version revision number.

The `totalLFSCounters` field contains the total number of LFS counters.

The `currentLFSCounters` field contains the number of LFS counters returned by this call to SSGetVolumeSwitchInfo.

The `counters` field contains the first LFS counter. This counter is followed by others.

You might want to define a structure to retrieve the LFS counters. However, an example structure ( LFSCountersStructure) is included in NWSERVST.H so that the fields can be identified. In the future this structure might grow beyond a size that can be retrieved with a single call to a function. The LFSCountersStructure is defined as follows:

```
typedef struct LFSCountersStructure
{
   LONG   ReadFile;
   LONG   WriteFile;
   LONG   DeleteFile;
   LONG   RenMove;
   LONG   OpenFile;
   LONG   CreateFile;
   LONG   CreateAndOpenFile;
   LONG   CloseFile;
   LONG   ScanDeleteFile;
   LONG   SalvageFile;
   LONG   PurgeFile;
   LONG   MigrateFile;
   LONG   DeMigrateFile;
   LONG   CreateDir;
   LONG   DeleteDir;
   LONG   DirectoryScans;
   LONG   MapPathToDirNum;
   LONG   ModifyDirEntry;
   LONG   GetAccessRights;
   LONG   GetAccessRightsFromIDs;
   LONG   MapDirNumToPath;
   LONG   GetEntryFromPathStrBase;
   LONG   GetOtherNSEntry;
   LONG   GetExtDirInfo;
   LONG   GetParentDirNum;
   LONG   AddTrusteeR;
   LONG   ScanTrusteeR;
   LONG   DelTrusteeR;
```

```
LONG   PurgeTrust;
LONG   FindNextTrustRef;
LONG   ScanUserRestNodes;
LONG   AddUserRest;
LONG   DeleteUserRest;
LONG   RtnDirSpaceRest;
LONG   GetActualAvailDskSp;
LONG   CntOwnedFilesAndDirs;
LONG   MigFileInfo;
LONG   VolMigInfo;
LONG   ReadMigFileData;
LONG   GetVolusageStats;
LONG   GetActualVolUsageStats;
LONG   GetDirUsageStats;
LONG   NMFileReadsCount;
LONG   NMFileWritesCount;
LONG   MapPathToDirectoryNumberOrPhantom;
LONG   StationHasAccessRightsGrantedBelow;
LONG   GetDataStreamLengthsFromPathStringBase;
LONG   CheckAndGetDirectoryEntry;
LONG   GetDeletedEntry;
LONG   GetOriginalNameSpace;
LONG   GetActualFileSize;
LONG   VerifyNameSpaceNumber;
LONG   VerifyDataStreamNumber;
LONG   CheckVolumeNumber;
LONG   CommitFile;
LONG   VMGetDirectoryEntry;
LONG   CreateDMFileEntry;
LONG   RenameNameSpaceEntry;
LONG   LogFile;
LONG   ReleaseFile;
LONG   ClearFile;
LONG   SetVolumeFlag;
LONG   ClearVolumeFlag;
LONG   GetOriginalInfo;
LONG   CreateMigratedDir;
LONG   F3OpenCreate;
LONG   F3InitFileSearch;
LONG   F3ContinueFileSearch;
LONG   F3RenameFile;
LONG   F3ScanForTrustees;
LONG   F3ObtainFileInfo;
LONG   F3ModifyInfo;
LONG   F3EraseFile;
LONG   F3SetDirHandle;
LONG   F3AddTrustees;
LONG   F3DeleteTrustees;
LONG   F3AllocDirHandle;
LONG   F3ScanSalvagedFiles;
LONG   F3RecoverSalvagedFiles;
LONG   F3PurgeSalvageableFile;
LONG   F3GetNSSpecificInfo;
LONG   F3ModifyNSSpecificInfo;
```

```
   LONG   F3SearchSet;
   LONG   F3GetDirBase;
   LONG   F3QueryNameSpaceInfo;
   LONG   F3GetNameSpaceList;
   LONG   F3GetHugeInfo;
   LONG   F3SetHugeInfo;
   LONG   F3GetFullPathString;
   LONG   F3GetEffectiveDirectoryRights;
   LONG   ParseTree;
}LFSCountersStructure;
```

Each field in this structure indicates the number of times a given OS code path is taken.

# Server-Based TTS Concepts

# 12

This documentation provides and overview of Server-Based TTS Services concepts, its functions, and features

The NetWare® Transaction Tracking System (TTS) feature ensures data integrity on files that otherwise would be corrupted when updates on the files are interrupted by such things as hardware failures or power outages. A transaction is defined as a set of one or more operations that must be completed together to maintain file and database integrity. TTS guarantees that all writes within a transaction are completed or none are completed.

A banking database application frequently performs a transaction that includes three writes to database files: a debit to one account, a credit to another account, and a note to a log. The application must complete all three writes to maintain database integrity. However, hardware or power failure can interrupt such a transaction, causing database corruption.

Servers running NetWare 3.x and above can track transactions and ensure file integrity by backing out (or erasing) interrupted or partially completed transactions. TTS tracks only transactions to transactional files. A file becomes transactional when the files transactional extended file attribute is set.

## 12.1 Transaction Process

The following steps describe how TTS tracks each write within a transaction:

1. An application writes new data to a file on a server.
2. The server stores the new data in cache memory. The target file on the servers hard disk remains unchanged.
3. The server scans the target file on the servers hard disk, finds the data to be changed (old data), and copies the old data to cache memory. The server also records the name and directory path of the target file and the location and length of the old data (record) within the file. The target file on the servers hard disk remains unchanged.
4. The server writes the old data in cache memory to a transaction work file on the servers hard disk. The transaction work file resides at the root level of the SYS volume on the server. The file is flagged System and Hidden. The target file on the servers hard disk remains unchanged.
5. The server writes the new data in cache memory to the target file on the servers hard disk. The target file is now changed.

The server repeats these steps for each write within a transaction. The transaction work file grows to accommodate the old data for each write. If the transaction is interrupted, the server writes the contents of the transaction work file to the target file, thereby restoring the file to pretransaction condition. In effect, the server backs out the transaction.

A server can monitor from 1 to 10,000 transactions. The maximum value is configurable and can be set from 1 to 10,000. A server can track only one transaction at a time for each unique connection number/task number pair.

# 12.2 Transaction Tracking

Transactions are divided into two categories:


- "Implicit Transaction Tracking" on page 592
- "Explicit Transaction Tracking" on page 593


Implicit transaction tracking requires no coding on the part of an application developer. If TTS is enabled on a server, TTS tracks all transactions to all transactional files (including transactions made by NetWare to Bindery files).

Explicit transaction tracking requires applications to make TTS calls and allows applications to neatly bracket file update sequences with locking and TTS calls. An application would most likely use logical or physical record locks with TTS functions. Almost all applications should work correctly with TTS implicit or explicit transactions, or be easily adapted to do so. However, the following TTS record locking features could adversely affect certain applications:

- Physical and logical record locks remain in force until the end of the transaction.
- Records unlocked in the middle of a transaction are usually held until the transaction completes.
- The server automatically generates a physical lock when a record is written if the record is not yet locked.

---

**NOTE:** Transactions are tracked and evaluated by connection number and task number. Therefore, in order for transactions on a file to be monitored, the thread must have its connection and task numbers set to the same connection and task numbers that were used to open the file. (Remember, setting the connection and task number for one thread changes the connection and task numbers for all of the threads that belong to the same thread group.)

---

## 12.2.1 Implicit Transaction Tracking

Implicit transaction tracking is designed to work transparently with existing multiuser software that uses record locking (physical or logical). All the user must do is flag the multiuser database files as transactional; everything else is automatic. However, implicit transactions are not guaranteed to work with all multiuser applications. For example, applications that do not synchronize file updates exclusively with record locks and applications that synchronize updates improperly might not work.

Applications that always keep one or more records locked should set lock thresholds:

- Application thresholds are valid only until the next end-of-job on the current connection. When the next end-of-job occurs (normally when a workstation application ends), the application thresholds are set to the current workstation thresholds. This is meaningful only in a client-server NLM application that uses the clients connection number for transactions. Application thresholds can be set using TTSSetApplicationThresholds.
- Workstation thresholds are valid not only for the requesting application but for all applications at the requesting workstation and all NLM applications using the same connection number as the callers current connection number. The threshold values are not reset when the application terminates (that is, when an end-of-job occurs). Workstation thresholds can be set using TTSSetWorkstationThresholds. Workstation users normally execute SETTTS.EXE to set workstation thresholds.

### 12.2.2 Explicit Transaction Tracking

Explicit transaction tracking has an advantage over implicit transaction tracking in that it allows applications to determine precisely when updates within the transaction are written to disk.

In addition, TTSBeginTransaction and TTSEndTransaction within the application allow the developer to identify the beginning and end-of-file update sequences. Identifying the beginning and end-of-file update sequences makes it easier for applications to group file updates and practice correct record locking practices. Correct record locking practices include locking and unlocking records as a group to avoid deadlock, and not leaving records locked exclusively while waiting for keyboard input.

## 12.3 Record Locking

Record locking provides security and data protection during transactions. If a record is not locked by an application but is written to during a transaction, the server physically locks that record automatically. This physical lock remains in force until the transaction is completed. Physically locking written records is an added protection that prevents other NLM applications and workstations from examining or modifying them while they are being Changed. The server usually holds logical and physical record locks on a file until the end of the transaction, even if the file is unlocked by an NLM before the transaction is completed.

For example, if an NLM requests an unlock before a transaction is completed, it receives an ESUCCESS completion code indicating that the records are unlocked; however, the lock is actually held until the transaction is complete. The one exception to this is a file that is not updated while the lock is in force. A request to unlock such a record is honored.

The server delays record unlocking because the data controlled by the locks could still be changed by a transaction backout. Thus, the server guarantees that other NLM applications and workstations do not see data that is being changed until those changes are final. If multiple stations attempt to change a file, only the first station is allowed to make the change. Usually, multiuser software synchronizes the records and prevents other NLM applications and workstations from examining records that are being changed.

If an NLM or workstation attempts to read from or write to a record that is physically locked by the server, the NLM or workstation gets a locked error. This means that applications that use logical record locks can potentially get unexpected physical lock errors. However, because unlocking logical records is also delayed during a file write, unexpected errors should never occur if the logical record synchronization is correct. The logical record locks keep other workstations away from the physically locked records.

It is important to point out that in a environment that is not TTS it is valid to unlock some updated records in the middle of a transaction if they have been completely updated. In a TTS environment, however, those records cannot be unlocked because they can still be changed-not by the application but by a transaction backout.

## 12.4 Transaction Backouts

Transactions are backed out because of system failures resulting from hardware problems and power outages at a workstation or the server. But backouts also occur because of problems with applications running on a workstation, or because of user intervention at a workstation:


- “Causes of Transaction Backout” on page 594

• "Solutions for Transaction Backout" on page 594

## 12.4.1 Causes of Transaction Backout

Some common causes for a transaction backout are listed below:

- An application terminates while a transaction is in progress (a begin transaction with no end transaction). For example, the user may enter a Ctrl+C.
- After terminating, a workstation application leaves records locked.
- There has been no activity from a station for 15 minutes. If a server does not receive packets from one of the workstations listed in its File Server ID Table for more than 5 minutes, it sends a watchdog packet to that station. If the station does not respond, the server continues to send a packet every minute for 15 minutes.
- After 15 minutes, if the workstation still does not respond, the server logs it out. Usually lack of response from a workstation indicates a power failure, software hang, a problem with an intermediate network route, and so on.
- A station reattaches to the same server after rebooting and reloading the shell. If the station attaches to a different server after a reboot, the server watchdog process reinitializes the station after 5 minutes.
- A CLEAR STATION or a DOWN command is issued at the server system console.
- The server crashes due to a hardware, software, or power problem.

## 12.4.2 Solutions for Transaction Backout

The following are some general suggestions of how to recover when a workstation or server that goes down and therefore backs out a transaction.

### Workstation

When a workstation goes down in the middle of a transaction, the user may not know exactly which transaction was completed and which was backed out. However, unless the application is multitasking, it should have only one transaction active at the time of the crash. Even if a workstation goes down, transactions that were completed but not yet written to disk are written to disk and not backed out.

If the station goes down and transactions were backed out, the user must ensure that the last transaction was completed, or if it was not completed, do it over again. The transaction could be partially completed if the application incorrectly used several transactions per operation. The following message appears at the server console if transactions were backed out:

```
Transaction being backed out for station ##.
```

This message is given at the server console if a transaction is backed out for any reason other than a TTSAbortTransaction from the application. Note that if the station had several tasks active, several messages are given. The DOWN server console command can force several of these messages to be issued.

If a workstation goes down and cannot be brought back up, the user should issue a CLEAR STATION command at the server console. Besides affecting TTS, the downed station keeps its records locked, which may affect other stations. If the CLEAR STATION command is not issued, the server watchdog process clears out the station after 15 minutes.

### Server

Unlike the workstation, if the server goes down before transactions have been written to the disk, it backs out any incomplete transactions when it is rebooted. If your server does go down and files are backed out, a message is issued at the server console indicating the number of transactions that need to be backed out.

This message appears when the server is being brought up (while the affected volumes are being mounted). The transaction tracking software has detected that some of the transactions were not completed before the server went down. You should not see this message *unless the server went down abnormally*.

After bringing up a server that went down, the user is responsible for determining which updates were completed and which were backed out and need to be repeated. A good practice is to verify that the last few changes were actually made. Remember that most transactions should be committed to disk within 3 to 5 seconds of being completed. However, any operation completed within 10 seconds of a failure should be examined. After the server is brought up, a good practice is to have the workstation operators review their last operation. If it is not completed, they should also review the next to last operation, and so on.

If TTSTransactionStatus is used by the application, the user has a better idea of when transactions were actually completed. However, the fact that the application thought that a transaction was not completed does not mean that it was not. The transaction could have been completed between the last time the application checked and the time the server crashed.

# 12.5 Disable/Enable Transaction Tracking

Transaction tracking can be disabled or enabled from the server console with the commands Disable Transactions and Enable Transactions. However, even when transaction tracking is disabled, tracking occurs in a partial way:

- TTSBeginTransaction and TTSEndTransaction still denote transactions and work correctly.
- TTSTransactionStatus still returns the correct status.

If TTS is disabled, the only loss to explicit and implicit transaction calls is the capability of backing out incomplete transactions. For example, if a server or workstation fails while transaction tracking is disabled, you cannot back out because the backout information was not saved, and the database could be corrupted.

When transaction tracking is re-enabled at the system console, backout capability is provided only to new transactions. Transactions that are in progress when transaction tracking is reenabled do not have backout, and the possibility of file corruption still exists until these unprotected transactions are completed and written to disk. It is important to note that a transaction can be backed out if the workstation or NLM does not write to a transaction file after transactions are disabled. If a workstation or NLM does a write after transactions are disabled, the transaction is invalidated and any previous backout information for that transaction is ignored.

## 12.5.1 Disable Transactions

Disable Transactions is a server console command that turns off TTSs backout capability. Any transactions written after transactions have been disabled cannot be backed out. A transaction that has not written anything since transactions were disabled can still be backed out while transactions are disabled. Normally, this command is used only for testing.

### 12.5.2 Enable Transactions

Enable Transactions is a server console command that re-enables TTS. Any previous transaction backout information is erased, except those that were active at the time transactions were disabled and did not write anything. (These transactions are not erased and can still be backed out.)

## 12.6 Functions

The following table lists the Transaction Tracking System Functions

| | |
|---|---|
| TTSAbortTransaction | Aborts explicit and implicit transactions |
| TTSBeginTransaction | Begins an explicit transaction |
| TTSEndTransaction | Ends an explicit or implicit transaction and returns a transaction reference number |
| TTSGetApplicationThresholds | Returns the current connections application thresholds for implicit transactions |
| TTSGetWorkstationThresholds | Returns the current connections workstation thresholds for implicit transactions |
| TTSIsAvailable | Verifies whether the default server supports transaction tracking |
| TTSSetApplicationThresholds | Sets the current connections application thresholds for implicit transactions |
| TTSSetWorkstationThresholds | Sets the current connections workstation thresholds for implicit transactions |
| TTSTransactionStatus | Verifies whether a transaction has been written to disk |

# Server-Based TTS Functions

# 13

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

# TTSAbortTransaction

Aborts explicit transactions (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWTTSAbortTransaction (page 367).)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based TTS

## Syntax

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

int TTSAbortTransaction (void);
```

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 253 | (0xFD) | ERR_TTS_DISABLED |
| | | Transaction could not be ended because the Transaction Tracking System™ (TTS™) is disabled. |
| 254 | (0xFE) | ERR_TRANSACTION_ENDS_RECORDS_LOCKED |
| | | Transaction was ended abnormally, but records were left locked. |
| 255 | (0xFF) | ERR_NO_EXPLICIT_TRANSACTION_ACTIVE |

## Remarks

When this function returns, the transaction has been successfully backed out. When a transaction is backed out, any writes to the file are aborted, and the file is returned to the state it was in before the transaction began.

This function releases the following physical record locks:

- Physical record locks generated by TTS when a write operation is performed on a nonlocked record
- Physical record locks that are not yet released because of a file write

For explicit transactions, the transaction is backed out, and any locks being held are released.

If implicit transactions are enabled and the number of logical or physical records still locked by the application exceeds the threshold, this function returns a value of 0xFE. In this case, the transaction is backed out, and the server automatically starts a new implicit transaction.

## See Also

TTSBeginTransaction (page 600), TTSEndTransaction (page 602), TTSIsAvailable (page 608), TTSTransactionStatus (page 613)

# TTSBeginTransaction

Begins an explicit transaction (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call NWTTSBeginTransaction (page 369).)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based TTS

## Syntax

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

int TTSBeginTransaction (void);
```

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 150 | (0x96) | ERR_SERVER_OUT_OF_MEMORY |
| 254 | (0xFE) | ERR_IMPLICIT_TRANSACTION_ACTIVE<br>Converted to an explicit transaction. |
| 255 | (0xFF) | ERR_EXPLICIT_TRANSACTION_ACTIVE |

## Remarks

After TTSBeginTransaction is called, TTS tracks all files that are marked transactional and are currently open as well as transactional files that are opened during the transaction. When an application writes to a transaction file, the server automatically generates a physical record lock for that file. If the record is already locked, the server does not generate an additional lock.

Transaction files are not closed and unlocked until a TTSEndTransaction or TTSAbortTransaction is executed.

If transaction files are updated, logical and physical record locks are held until the end of the transaction. If a transaction file is not updated, any logical or physical lock on that file can be released at any time.

**NOTE:** Transactions are tracked and evaluated by connection number and task number. Therefore, in order for transactions on a file to be monitored, the thread must have its connection and task numbers set to the same connection and task numbers that were used to open the file. (Remember,

setting the connection and task number for one thread changes the connection and task numbers for all of the threads that belong to the same thread group.)

## See Also

TTSAbortTransaction (page 598), TTSEndTransaction (page 602), TTSIsAvailable (page 608), TTSTransactionStatus (page 613)

# TTSEndTransaction

Ends an explicit transaction and returns a transaction reference number (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call call NWTTSEndTransaction (page 371).)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based TTS

## Syntax

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

int TTSEndTransaction (
   long   *transactionNumber);
```

## Parameters

**`transactionNumber`**

(OUT) Receives the transaction reference number of the ended transaction.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 253 | (0xFD) | ERR_TTS_DISABLED |
| | | Transaction could not be ended because TTS is disabled. |
| 254 | (0xFE) | ERR_TRANSACTION_ENDS_RECORDS_LOCKED |
| | | Transaction was ended abnormally, but records were left locked. This error is only for implicit transactions. |
| 255 | (0xFF) | ERR_EXPLICIT_TRANSACTION_ACTIVE |

## Remarks

A transaction is not necessarily completed to disk when this function returns. If the server fails before all transaction updates are written to disk, the transaction is backed out when the server is rebooted.

TTSTransactionStatus verifies a successful transaction completion to disk; or, if transaction tracking is disabled, the reference number can used to determine when the transaction is completely written to disk.

This function releases the following physical record locks:

- Physical record locks generated by TTS when a write operation is performed on a nonlocked record
- Physical or logical locks that have not yet been released because of a transaction

For explicit transactions, the transaction is completed and any locks being held are released.

If implicit transactions are enabled and the number of logical or physical records still locked by the application exceeds the threshold, this function returns a value of 0xFE. In this case, the server automatically starts a new implicit transaction. However, the explicit transaction is still completed and any locks being held are released.

## See Also

TTSAbortTransaction (page 598), TTSBeginTransaction (page 600), TTSIsAvailable (page 608), TTSTransactionStatus (page 613)

# TTSGetApplicationThresholds

Returns the current connection's application thresholds for implicit transactions (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call call NWTTSGetConnectionThresholds (page 373).)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based TTS

## Syntax

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

int TTSGetApplicationThresholds (
   BYTE   *logicalRecordLockThreshold,
   BYTE   *physicalRecordLockThreshold);
```

## Parameters

**logicalRecordLockThreshold**

(OUT) Receives the number of logical record locks allowed before an implicit transaction begins (0 to 255).

**physicalRecordLockThreshold**

(OUT) Receives the number of physical record locks allowed before an implicit transaction begins (0 to 255).

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| NetWare® Error | | UNSUCCESSFUL |

## Remarks

This function allows an application to get the number of logical and physical record locks allowed before an implicit transaction begins for the current connection.

This function and the TTSSetApplicationThresholds function are useful for applications that change the implicit application thresholds and later want to restore them.

For example, TTSGetApplicationThresholds can query an application for the number of logical and physical record locks allowed before an implicit transaction begins.

TTSSetApplicationThresholds can then do one of the following:

- Turn off implicit transactions.
- Set implicit thresholds for applications that always keep one or more records locked.

Applications that intend to use only explicit transactions, but sometimes generate unnecessary implicit transactions, would need to turn off all implicit transactions.

The default threshold for logical and physical locks is 0. A threshold of 0xFF means there are not implicit transactions for that lock type.

The thresholds returned by this function are valid for the requesting application only. When the application terminates, the workstation thresholds are restored. This is only meaningful in a client-server NLM™ application that uses the client's connection number for transactions. Whenever the client's workstation generates an end-of-job (normally when a workstation application ends), the TTS application thresholds are reset to the TTS workstation thresholds.

## See Also

TTSGetWorkstationThresholds (page 606), TTSSetApplicationThresholds (page 609), TTSSetWorkstationThresholds (page 611)

# TTSGetWorkstationThresholds

Returns the current connection's workstation thresholds for implicit transactions

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based TTS

## Syntax

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

int TTSGetWorkstationThresholds (
   BYTE   *logicalRecordLockThreshold,
   BYTE   *physicalRecordLockThreshold);
```

## Parameters

**logicalRecordLockThreshold**

(OUT) Specifies the number of logical record locks allowed before an implicit transaction begins (0 to 255).

**physicalRecordLockThreshold**

(OUT) Specifies the number of physical record locks allowed before an implicit transaction begins (0 to 255).

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| NetWare Error | | UNSUCCESSFUL |

## Remarks

This function allows an application to get the number of logical and physical record locks allowed before an implicit transaction begins for the current connection.

This function and TTSSetWorkstationThresholds are useful for applications that want to set the thresholds semi-permanently (until the next time TTSetWorkstationThresholds is called).

For example, TTSGetWorkstationThresholds can get the number of logical and physical locks, and TTSSetWorkstationThresholds can do one of the following:

- Turn off implicit transactions.

- Set implicit thresholds for applications that always keep one or more records locked.

Applications that use only explicit transactions, but sometimes generate unnecessary implicit transactions, can turn off all implicit transactions.

The default threshold for logical and physical locks is 0. A threshold of 0xFF means there are no implicit transactions for that lock type.

The threshold values set by TTSSetWorkstationThresholds are not reset when a workstation application terminates (that is, when an end-of-job occurs). In other words, all applications on the requesting workstation share these same thresholds.

Workstation users normally execute SETTTS.EXE to set workstation thresholds.

## See Also

TTSSetWorkstationThresholds (page 611)

# TTSIsAvailable

Reports whether the server has transaction tracking enabled (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call call NWTTSIsAvailable (page 379).)

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based TTS

## Syntax

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

int TTSIsAvailable (void);
```

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 1 | (0x01) | TTS_AVAILABLE |
| 253 | (0xFD) | ERR_TTS_DISABLED: TTS is available but is presently disabled. |

## Remarks

The NetWare 3.x and 4.x OS always supports TTS, but TTS can be disabled.

# TTSSetApplicationThresholds

Sets the current connection's application thresholds for implicit transactions (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call and call NWTTSSetConnectionThresholds (page 381).)

**Local Servers:** nonblocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based TTS

## Syntax

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

int TTSSetApplicationThresholds (
   BYTE   logicalRecordLockThreshold,
   BYTE   physicalRecordLockThreshold);
```

## Parameters

**logicalRecordLockThreshold**

(IN) Specifies the maximum number of logical locks that can be held without starting a transaction.

**physicalRecordLockThreshold**

(IN) Specifies the maximum number of physical locks that can be held without starting a transaction.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| NetWare Error | | UNSUCCESSFUL |

## Remarks

The thresholds set by this function are valid only until the next end-of-job on the current connection. When the next end-of-job occurs, the application thresholds are set to the current workstation thresholds.

This function is useful for either turning off implicit transactions or allowing applications to work that always keep one or more records locked. For example, applications that use only explicit

transactions, but sometimes generate unnecessary implicit transactions, can use this function to turn off all implicit transactions.

The default threshold for logical and physical locks is 0. A threshold of 0xFF means there are no implicit transactions for that lock type.

The thresholds set by this function are valid for the requesting application only. When the application terminates, the workstation thresholds are restored. This is only meaningful in a client-server NLM that uses the client's connection number for transactions. Whenever the client's workstation generates an end-of-job (normally when a workstation application ends), the TTS application thresholds are reset to the TTS workstation thresholds.

## See Also

TTSGetApplicationThresholds (page 604)

# TTSSetWorkstationThresholds

Sets the current connection's workstation thresholds for implicit transactions

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based TTS

## Syntax

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

int TTSSetWorkstationThresholds (
   BYTE   logicalRecordLockThreshold,
   BYTE   physicalRecordLockThreshold);
```

## Parameters

**logicalRecordLockThreshold**

(IN) Specifies the number of logical record locks allowed before an implicit transaction begins (0 to 255).

**physicalRecordLockThreshold**

(IN) Specifies the number of physical record locks allowed before an implicit transaction begins (0 to 255).

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| NetWare Error | | UNSUCCESSFUL |

## Remarks

The thresholds set by this function are valid not only for the requesting application, but for all applications at the requesting workstation and for all NLM applications using the same connection number as the caller's current connection number. This function is useful for either turning off implicit transactions or allowing applications to work that always keep one or more records locked.

For example, applications that intend to use only explicit transactions but which sometimes generate unnecessary implicit transactions can use this function to turn off all implicit transactions.

The default threshold for logical and physical locks is 0. A threshold of 0xFF means there are no implicit transactions for that lock type.

The threshold values set by this function are not reset when a workstation application terminates. In other words, all applications on the requesting workstation share these thresholds.

Workstation users normally execute SETTTS.EXE to set workstation thresholds.

## See Also

TTSGetWorkstationThresholds (page 606)

# TTSTransactionStatus

Verifies whether a transaction has been written to disk (For cross-platform functionality, see Developing NLMs with Cross-Platform Functions ( *NDK: NLM Development Concepts, Tools, and Functions*) and call call NWTTSTransactionStatus (page 387).)

**Local Servers:** blocking

**Remote Servers:** blocking

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

**Platform:** NLM

**Service:** Server-Based TTS

## Syntax

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

int TTSTransactionStatus (
   long   transactionNumber);
```

## Parameters

**`transactionNumber`**

(IN) Specifies the transaction reference number returned from the TTSEndTransaction function.

## Return Values

| Decimal | Hex | Constant |
|---|---|---|
| 0 | (0x00) | ESUCCESS |
| 255 | (0xFF) | TRANSACTION_NOT_YET_WRITTEN |

## Remarks

This function verifies whether a transaction has been written to disk, even if TTSEndTransaction returned ERR_TTS_DISABLED.

This function uses the `transactionNumber` parameter that contains a number assigned and used by the NetWare operating system to track each transaction. The `transactionNumber` is obtained through a call to TTSEndTransaction.

Because of the server caching algorithms, it can be 3 to 5 seconds or longer before transactions are actually written. Transactions are completed to disk in the order in which they are ended.

## See Also

TTSEndTransaction (page 602)

# Revision History

# A

The following table outlines all the changes that have been made to the Server Management documentation (in reverse chronological order).

| | |
|---|---|
| March 1, 2006 | Updated format. |
| December 6, 2005 | Updated NWSMLoadNLM2 (page 344). |
| October 5, 2005 | Transitioned to revised Novell documentation standards. |
| March 2, 2005 | Fixed legal information. |
| June 9, 2004 | Added some information to VERSION_INFO (page 316) . |
| July 30, 2003 | Fixed the Delphi syntax for NWFSE_MLID_BOARD_INFO (page 263). |
| June 2003 | Added information to SSGetMediaManagerObjChildList (page 531). Added NLM Type information to NLM_INFO (page 223) and SSGetNLMInfo (page 548). Removed the obsolete label from the SS... functions. Changed references to Pascal to Delphi. Modified the Delphi syntax for NWFSE... structures to add padding. |
| October 2002 | Modified the Pascal syntax in the structures.<br><br>Marked all SS... functions as obsolete. Added additional information to SSGetFileServerInfo (page 495) and SSGetUserInfo (page 581). |
| September 2002 | Added Ethernet frame information to the MLID_BOARD_INFO (page 220) structure and tick information to the SERVER_AND_VCONSOLE_INFO (page 306) structure. |
| May 2002 | Added information about the CPUNum in NWGetCPUInfo (page 53) and the Pentium Pro value for CPUTypeFlag in CPU_INFO (page 182). |
| February 2002 | Added information about the netWareVersion and netwareSubVersion values to GetServerInformation (page 468).<br><br>Updated links. |
| October 2001 | Updated Pascal syntaxes for NWEnumNetAddresses (page 41), NWGetMLIDBoardInfo (page 110), NWEnumNetAddresses (page 41), NWGetServerConnInfo (page 143), MLID_BOARD_INFO (page 220), NWFSE_ACCT_INFO (page 226), NWFSE_AUTH_INFO (page 233), NWFSE_LOCK_INFO (page 253), NWFSE_LOGIN_NAME (page 254), NWFSE_LOGIN_TIME (page 255), NWFSE_NETWORK_ADDRESS (page 264), NWFSE_MLID_BOARD_INFO (page 263), NWFSE_PRINT_INFO (page 276), NWFSE_STATS_INFO (page 289), and VOLUME_INFO_BY_LEVEL (page 319).<br><br>Fixed typographical errors. |

| | |
|---|---|
| September 2001 | Added NetWare 6.0 support to documentation.<br><br>Updated NWGetServerConnInfo (page 143), SSGetMediaManagerObjList (page 539), NWFSE_AUTH_INFO (page 233), NWFSE_MEDIA_MGR_OBJ_LIST (page 260), and NWFSE_PRINT_INFO (page 276).<br><br>Fixed typographical errors.<br><br>Added descriptions to graphics. |
| June 2001 | Updated tables and fixed typographical errors. |
| February 2001 | Corrected #include syntax entry by adding nit\ in front of the nwservst.h file reference for all SSGet... functions in the Chapter 11, “Server-Based Server Environment Functions,” on page 395. Added Delphi (Pascal) syntax to functions where missing.<br><br>Changed the referenced structure in NWGetLANCommonCountersInfo (page 89) to NWFSE_LAN_COMMON_COUNTERS_INFO (page 248).<br><br>Added a note to NWFSE_NETWORK_ADDRESS (page 264) about the declaration of address types 1 and 2 being different from the standard NDS declaration of network address types. |
| July 2000 | Added documentation on the new function NWGetNetWareProductVersion (page 114) and the associated structure NETWARE_PRODUCT_VERSION (page 222). Also, updated documentation for the two functions: NWGetFileServerVersionInfo (page 74) and NWGetOSVersionInfo (page 128). |
| May 2000 | Changed numberOfRecords of CONN_USING_FILE_REPLY_386 in GetConnectionsUsingFile (page 427) from a BYTE to a WORD per the updated nlm/nit/nwenvrn1.h file.<br><br>Changed some field descriptions in FSE_MM_OBJ_INFO (page 198). |
| March 2000 | Added field definitions for FSE_FILE_SYSTEM_INFO (page 196), VOLUME_INFO_BY_LEVEL_DEF (page 320), and VOLUME_INFO_BY_LEVEL_DEF2 (page 325).<br><br>Moved Server-Based Auditing Concepts, Bindery-Based Accounting Concepts and Bindery-Based Accounting Functions to Network Management.<br><br>Changed fourth possible value of `retInfoMask` in NWGetServerConnInfo (page 143) to CONN_INFO_LOGIN_NAME_MASK. Also, changed the third parameter of NWFSE_LOGIN_NAME (page 254) to be a pointer.<br><br>Added statement to NWGetServerConnInfo (page 143) about having to allocate memory for `networkAddress` or the server might abend. |
| January 2000 | Updated the Remarks section of NWGetNLMLoadedList (page 124) to indicate the correct starting value of `startNum`.<br><br>Changed the corresponding cross-platform function for GetServerInformation (page 468) to NWGetFileServerVersionInfo (page 74).<br><br>Changed Return Values for TTSIsAvailable (page 608). |

| | |
|---|---|
| November 1999 | Added library information to each function.<br><br>Added Remarks statements in NWGetMLIDBoardInfo (page 110) about the limit of one protocol per board in NetWare 4.x and that 0x89FF is returned if no protocols are bound to the specified board.<br><br>Added possible protocol names, numbers, and IDs for MLID_BOARD_INFO (page 220). |
| September 1999 | Removed NetWare 3.x as a supported platform from NWSMLoadNLM (page 342) and NWSMUnloadNLM (page 353). These functions use NCP 131 1 and 131 2, which work only on NetWare 4.x and above.<br><br>Removed 0x009E as a return value for NWSMLoadNLM (page 342).<br><br>Changed `serverType` to an IN parameter in NWGetKnownServersInfo (page 87).<br><br>Changed `lastRecord` and `lastTask` to be IN/OUT parameters in GetConnectionSemaphores (page 415).<br><br>Changed the `addressType` values of NWFSE_NETWORK_ADDRESS (page 264) to 1 IPX, 2 IP, 8 UDP, and 9 TCP. |
| July 1999 | Removed NW_FS_INFO and NW_DYNAMIC_MEM structures, which were used only by NWGetFilServerMiscInfo (2.x only function). |
| June 1999 | Added NWGenerateGUIDs (page 43).<br><br>Added NWGetMLIDBoardInfo (page 110) and related NWFSE_MLID_BOARD_INFO (page 263) and MLID_BOARD_INFO (page 220) structures.<br><br>Added NWEnumNetAddresses (page 41) and related NW_GUID (page 294) structure.<br><br>Added NWGetServerConnInfo (page 143) and related NWFSE_ACCT_INFO (page 226), NWFSE_AUTH_INFO (page 233), NWFSE_LOCK_INFO (page 253), NWFSE_LOGIN_NAME (page 254), NWFSE_LOGIN_TIME (page 255), NWFSE_NETWORK_ADDRESS (page 264), NWFSE_PRINT_INFO (page 276), and NWFSE_STATS_INFO (page 289) structures. |