File Service Group

File Overview

AFP: Guides

Data Migration: Guides

Deleted File: Guides

Direct File System: Guides

DOS Partition: Guides

Extended Attribute: Guides

File System: Guides

File System Monitoring: Guides

Media Manager: Guides

Name Space: Guides

NetWare STREAMS: Guides

Operating System I/O: Guides

Path and Drive: Guides

Stream I/O: Guides

Synchronization: Guides

Volume: Guides

Volume Management: Guides

AFP

AFP: Guides

AFP: Task Guide

AFP Directories

Operating on AFP Directory Entries

AFP Support Information

Checking for AFP Support

Additional Links

AFP: Functions

AFP: Structures

Parent Topic:

AFP: Guides

AFP: Concept Guide

AFP File Information

AFP File Information

Accessing AFP File Information: Example

AFP Directories

AFP Entry IDs

Creating AFP Directory Entries: Example

AFP Support Information

Checking for AFP Support: Example

General AFP Information

AFP Filename and Path Conventions

Finder Information

AFP Data and Resource Forks

Additional Links

AFP: Functions

File Service Group

AFP: Structures

Parent Topic:

AFP: Guides

AFP: Tasks

Checking for AFP Support

NetWare® servers support AFP files on a volume-by-volume basis. However, this support is optional. The appropriate name space NLM™ application must be loaded at the server, and AFP support must be enabled on the volume. This is equivalent to asking if MAC.NAM (or the Macintosh namespace NLM) is loaded for the volume in question.

Before attempting to perform AFP operations on an entry, call **NWAFPSupported** to make sure the AFP name space is supported on the NetWare volume.

Parent Topic:

AFP: Guides

Operating on AFP Directory Entries

Always use AFP Services to create, delete, and rename AFP directory entries. The File Access Services used to access other NetWare® files cannot perform these operations since they are unable to preserve the relationship between the files data and resource forks.

AFP Services include the following functions to operate on AFP files:

NWAFPCreateDirectory

NWAFPCreateFile

NWAFPDelete

NWAFPOpenFileFork

NWAFPRename

These functions take a combination AFP entry ID and path string that identifies the entry. Also, the string should be length-preceded, the initial byte indicating the length of the string.

Parent Topic:

AFP: Guides

AFP: Examples

Accessing AFP File Information: Example

The following code uses **NWAFPGetFileInformation** to return the long name for an AFP file or directory. At the command line, type a full NetWare® path (including server and volume names). **NWParseNetWarePath** finds the server's connection handle. **NWAFPDirectoryEntry** verifies the path is an AFP directory entry.

NWParsePath and **NWGetVolumeNumber** are used together to find the volume number. In turn, the volume number is used as input to **NWAFPGetEntryIDFromPathName** to find the AFP entry ID. The example then prepares a request mask for returning the short and long names of the entry. The mask is passed to **NWAFPGetFileInformation**.

Getting an AFP Directory Name

```
/* ********************************************************************
 *
 * Name         : Getting an AFP Directory Name
 *
 *
 * Abstract     : Call NWAFPGetFileInformation to return the long name f
 *                AFP file or directory.
 *
 * Inputs       : Usage: AFPNAME <path>
 *
 * Outputs      : Short and long file names
 *
 * ********************************************************************
 */

#define AFP_FILE 1

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <nwcalls.h>
#include <nwdpath.h>
#include <nwafp.h>

void main(int argc, char *argv[])
{
      NWAFP_FILE_INFO      fileInfo;
      nuint32              AFPEntryID;
```

```
NWDIR_HANDLE          dirHandle;
nuint16               requestMask = 0;
NWCONN_HANDLE         conn;
nuint16               volNumber;
NWCCODE               ccode;
char                  volName[16];
char                  path[256];

if(argc < 2)
{     printf("Usage: AFPNAME <path>\n");
      exit(1);
}

ccode = NWCallsInit(NULL, NULL);
if(ccode)
      exit(1);

#ifndef N_PLAT_UNIX
strupr(argv[1]);
#endif

NWParseNetWarePath(argv[1], &conn, &dirHandle, path);

ccode = NWAFPDirectoryEntry(conn, dirHandle, path);
if(ccode != AFP_FILE)
      exit(1);

ccode = NWAFPGetEntryIDFromPathName(conn, dirHandle, path, &AFPEn
 if(ccode)
      exit(1);
requestMask = AFP_GET_LONG_NAME | AFP_GET_SHORT_NAME;

ccode = NWParsePath(path, NULL, NULL, volName, NULL);
if(ccode)
      exit(1);

ccode = NWGetVolumeNumber(conn, volName, &volNumber);
if(ccode)
      exit(1);

ccode = NWAFPGetFileInformation(conn, volNumber, AFPEntryID, requ
                                  sizeof(fileInfo), &fileInfo);
if(ccode)
      exit(1);

      printf("\nShort name: %s", fileInfo.shortName);
      printf("nLong name: %s", fileInfo.longName);
}
```

Parent Topic:

AFP: Guides

Checking for AFP Support: Example

The following code checks whether AFP is supported by volume SYS: (volume number 0) on a NetWare® server, SERV1.

AFP Support

```
/* ********************************************************************
 * Name         : Checking for AFP Support
 *
 * Abstract     : Call NWAFPSupported to make sure the AFP name space is

 *                on a NetWare volume.
 *
 * Notes        : This example checks whether AFP is supported by volume
 *                  SYS: (volume number 0) on server named SERV1.
 * ********************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <nwcalls.h>
#include <nwconnec.h>
#include <ntypes.h>

void main(void)
{
      NWCONN_HANDLE    conn;
      NWCCODE          ccode;

      ccode = NWCallsInit(NULL, NULL);
      if(ccode)
            exit(1);

      ccode = NWGetConnectionHandle((pnuint8)"SERV1", 0, &conn, NULL);
      if (ccode)
            exit(1);

      ccode = NWAFPSupported(conn, 0);  /* 0 = volume SYS: */
      if(ccode)
            printf("\nAFP not supported on volume SYS.\n");
      else
            printf("\nAFP supported on volume SYS.\n");
}
```

Parent Topic:

AFP: Guides

Creating AFP Directory Entries: Example

This code uses **NWAFPCreateDirectory** to create a directory with an AFP name. At the command line, type the directory path (including server and volume names) followed by the AFP directory name. Because the AFP name can contain spaces, the code checks for multiple arguments and concatenates them into a length-preceded long name.

NWParseNetWarePath parses the NetWare® path and returns the server's connection ID and a combination DOS directory handle and directory path. **NWParsePath** parses the resulting path and finds the volume name, which is used as input for **NWGetVolumeNumber**.

NWAFPGetEntryIDFromPathName returns an AFP entry ID for the DOS handle and path. **NWAFPCreateDirectory** creates the AFP directory. No Finder Information is passed to **NWAFPCreateDirectory**, so the Finder automatically creates that information when the entry is first accessed.

Creating AFP Directories

```
/* ********************************************************************
 *
 * Name         : Using AFP Directory Entry Operations
 *
 *
 * Abstract     : Call NWAFPCreateDirectory to create a directory with a
 *                AFP name.
 *
 *
 * Inputs       : Usage: AFPMD <directory path> AFP <directory name>
 *
 * ********************************************************************
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <nwcalls.h>
#include <ntypes.h>

void main (int argc, char *argv[ ])
{
      nuint32              entryID1, entryID2;
      nuint8               finderInfo;
      NWDIR_HANDLE         dirHandle;
      NWCONN_HANDLE        conn;
      nuint16              volNumber;
      NWCCODE              ccode;
      nstr8                afpPathString[32];
      nstr8                path[256] = "";
      nstr8                volName[16];
      nint16               i;

      if(argc < 3)
```

```
{     printf("Usage: AFPMD <server\path> <directory
            name>\n");
      exit(1);
}
strcpy(afpPathString + 1, argv[2]);
for(i = 3; i < argc; i++)
{      strcat(afpPathString + 1, " ");
      strcat(afpPathString + 1, argv[i]);
}
afpPathString[0] = strlen(afpPathString + 1);

ccode = NWCallsInit(NULL, NULL);
printf("\r\n NWCallsInit return ccode= %08lx",ccode);
if(ccode)
      exit(1);

#ifndef N_PLAT_UNIX
strupr(argv[1]);
#endif
NWParseNetWarePath(argv[1], &conn, &dirHandle, path);

ccode = NWParsePath(path, NULL, NULL, volName, NULL);
printf("\r\n NWParsePath return ccode= %08lx",ccode);
if(ccode)
      exit(1);

ccode = NWGetVolumeNumber(conn, volName, &volNumber);
printf("\r\n NWGetVolumeNumber ccode= %08lx",ccode);

ccode = NWAFPGetEntryIDFromPathName(conn, dirHandle, path, &entry
printf("\r\n NWAFPGetEntryIDFromPathName return ccode= %08lx",ccc
if(ccode)
      exit(1);

ccode = NWAFPCreateDirectory(conn, volNumber, entryID1, &finderIn
                                                  afpPathString, &en
printf("\r\n NWAFPCreateDirectory ccode= %08lx",ccode);
if(ccode)
{      printf("Unable to create \"%s\".\n", afpPathString + 1);
      exit(1);
}

printf("Directory \"%s\" created successfully.\n", afpPathString
}
```

Parent Topic:

AFP: Guides

AFP: Concepts

AFP Data and Resource Forks

AFP files are divided into a data fork and a resource fork. The data fork stores data formatted according to the creator's discretion. The resource fork, if present, stores data understood in prescribed formats, such as code, icons, menu bars, alerts, version information, and execution behavior.

From the AFP standpoint, a DOS file is a data file with no resource fork or long name. To endow a DOS file with a Macintosh name, the NetWare® OS permits Macintosh users to give the file a name in the Macintosh namespace. Otherwise, Macintosh users see the DOS name just as DOS users do.

Parent Topic:

AFP: Guides

AFP Entry IDs

The AFP entry ID is similar to the NetWare® directory handle. The AFP long filename relates to an AFP entry ID that represents some portion of the file's directory path. However, AFP and NetWare conventions are not interchangeable. Never mix NetWare directory handles with long names or AFP entry IDs with short names.

AFP Services includes the following functions for returning AFP entry IDs using a NetWare directory handle, a long name, or a short name.

NWAFPGetEntryIDFromHandle

NWAFPGetEntryIDFromName

NWAFPGetEntryIDFromPathName

Parent Topic:

AFP: Guides

AFP File Information

AFP Services include three functions that access AFP file information:

NWAFPGetFileInformation

NWAFPScanFileInformation

NWAFPSetFileInformation

NWAFPGetFileInformation and **NWAFPScanFileInformation** return file information in an NW_AFP_FILE_INFO structure.
NWAFPSetFileInformation uses an NW_AFP_SET_INFO structure to modify file information. AFP file information includes the following items:

The AFP ID of the entry

The AFP ID of the parent of the entry

File or directory attributes

Data fork size

Resource fork size

Number of files and subdirectories contained in the entry

Dates and times the entry was created, accessed, modified, and backed up

Macintosh Finder information

AFP long name

The object ID that created or last modified the entry

NetWare® name in the DOS or primary name space

Access privileges of the client

Apple Pro DOS information (Apple II Information)

All three functions include a request mask parameter that indicates the information to be returned or modified. The request mask defines the following values:

First Byte:

Bit 0 = AFP_GET_ATTRIBUTES
Bit 1 = AFP_GET_PARENT_ID
Bit 2 = AFP_GET_CREATE_DATE
Bit 3 = AFP_GET_ACCESS_DATE
Bit 4 = AFP_GET_MODIFY_DATE/TIME
Bit 5 = AFP_GET_BACKUP_DATE/TIME
Bit 6 = AFP_GET_FINDER_INFO
Bit 7 = AFP_GET_LONG_NAME

Second Byte:

Bit 0 = AFP_GET_ENTRY_ID
Bit 1 = AFP_GET_DATE_FORK_LEN
Bit 2 = AFP_RESOURCE_LEN
Bit 3 = AFP_GET_NUM_OFFSPRING
Bit 4 = AFP_GET_OWNER_ID
Bit 5 = AFP_GET_SHORT_NAME
Bit 6 = AFP_GET_ACCESS_RIGHTS
Bit 7 = undefined

Parent Topic:

AFP: Guides

AFP Filename and Path Conventions

AFP directories and files differ from their counterparts in NetWare® in the length of file names and the way naming paths are designated. AFP names can contain from 1 to 31 characters comprised of any ASCII character between 1 and 255 except the colon (:) or the NULL character. A NetWare server automatically generates short names (DOS-style filenames) for all AFP directories, as well as for any files created or accessed from DOS. The server maintains both the long name and the short name for each AFP directory and file.

Be aware that although most AFP functions use AFP directory paths, some require a directory path in NetWare format, and that AFP and NetWare formats cannot be mixed for entry. These differences are noted for each function in AFP: Functions.

Parent Topic:

AFP: Guides

Finder Information

The Mac OS uses Finder information---the file type, the icon's location in its parents window, and assorted file flags---to display files on the desktop. Operations such as creating an AFP file or directory require Finder information. If you pass a NULL value for the Finder information when you create a file, the Mac OS automatically creates Finder information.

Parent Topic:

AFP: Guides

AFP: Functions

NWAFPAllocTemporaryDirHandle

Allocates a directory handle for an AFP directory
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPAllocTemporaryDirHandle (
   NWCONN_HANDLE         conn,
   nuint16               volNum,
   nuint32               AFPEntryID,
   pnstr8                AFPPathString,
   NWDIR_HANDLE N_FAR   *dirHandle,
   pnuint8               accessRights);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPAllocTemporaryDirHandle
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   AFPEntryID : nuint32;
   AFPPathString : pnstr8;
   Var dirHandle : NWDIR_HANDLE;
   accessRights : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

volNum

(IN) Specifies the volume number of the directory entry location.

AFPEntryID

(IN) Specifies the AFP base ID.

AFPPathString

(IN) Points to the AFP style directory path relative to *AFPEntryID*.

dirHandle

(OUT) Points to the NetWare directory handle.

accessRights

(OUT) Points to the effective rights the requesting user has on the directory.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8805	NET_RECV_ERROR
0x8988	INVALID_FILE_HANDLE
0x8983	IO_ERROR_NETWORK_DISK
0x8993	NO_READ_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899D	NO_MORE_DIRECTORY_HANDLES
0x89A1	DIRECTORY_IO_ERROR

Remarks

The directory handles allocated by **NWAFPAllocTemporaryDirHandle** are automatically deallocated when the task terminates.

NCP Calls

0x2222 35 11 AFP Alloc Temporary Directory Handle

See Also

NWAllocTemporaryDirectoryHandle, **NWAllocTempNSDirHandle2**

NWAFPASCIIZToLenStr

Changes a NULL-terminated string to a length-preceded string
NetWare Server:
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

```
#include <nwcalls.h>

NWCCODE NWAPI NWAFPASCIIZToLenStr (
   pnstr8   dstStr,
   pnstr8   srcStr);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPASCIIZToLenStr
  (pbstrDstStr : pnstr8;
   pbstrSrcStr : pnstr8
) : NWCCODE;
```

Parameters

dstStr

(OUT) Points to a length-preceeded string of PASCAL type.

srcStr

(IN) Points to a NULL-terminated string.

Return Values

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

0x0000	SUCCESSFUL

Remarks

NWAFPASCIIZToLenStr returns the length of the string if it is greater than the predetermined accepted size.

NCP Calls

None

NWAFPCreateDirectory

Creates an AFP directory
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPCreateDirectory (
   NWCONN_HANDLE   conn,
   nuint16         volNum,
   nuint32         AFPEntryID,
   pnuint8         finderInfo,
   pnstr8          AFPPathString,
   pnuint32        newAFPEntryID);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPCreateDirectory
 (conn : NWCONN_HANDLE;
  volNum : nuint16;
  AFPEntryID : nuint32;
  finderInfo : pnuint8;
  AFPPathString : pnstr8;
  newAFPEntryID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number of the directory entry location.

AFPEntryID

(IN) Specifies the AFP base ID.

finderInfo

(IN) Points to AFPFILEINFO containing the finder information for the new directory.

AFPPathString

(IN) Points to the AFP directory path relative to *AFPEntryID*.

newAFPEntryID

(OUT) Points to the ID of the newly created directory.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8805	NET_RECV_ERROR
0x8983	IO_ERROR_NETWORK_DISK
0x8984	NO_CREATE_PRIVILEGES
0x8988	INVALID_FILE_HANDLE
0x8993	NO_READ_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899C	INVALID_PATH
0x899E	INVALID_FILENAME
0x89A1	DIRECTORY_IO_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER

NCP Calls

0x2222 35 13 AFP 2.0 Create Directory

See Also

NWAFPDelete, NWCreateDirectory

NWAFPCreateFile

Creates an AFP file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPCreateFile (
   NWCONN_HANDLE   conn,
   nuint16         volNum,
   nuint32         AFPEntryID,
   nuint8          delExistingFile,
   pnuint8         finderInfo,
   pnstr8          AFPPathString,
   pnuint32        newAFPEntryID);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPCreateFile
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   AFPEntryID : nuint32;
   delExistingFile : nuint8;
   finderInfo : pnuint8;
   AFPPathString : pnstr8;
   newAFPEntryID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number of the directory's entry location.

AFPEntryID

(IN) Specifies the AFP base ID.

delExistingFile

(IN) Specifies whether to delete the file of the same name (0 = do not delete).

finderInfo

(IN) Points to AFPFILEINFO containing the finder information for the new file.

AFPPathString

(IN) Points to the AFP directory path relative to *AFPEntryID*.

newAFPEntryID

(OUT) Points to the ID of the newly created directory.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8805	NET_RECV_ERROR
0x8980	ERR_LOCK_FAIL
0x8981	NO_MORE_FILE_HANDLES
0x8983	IO_ERROR_NETWORK_DISK
0x8984	NO_CREATE_PRIVILEGES
0x8987	WILD_CARDS_IN_CREATE_FILE_NAME
0x8988	INVALID_FILE_HANDLE
0x898A	NO_DELETE_PRIVILEGES
0x898D	SOME_FILES_AFFECTED_IN_USE
0x898E	NO_FILES_AFFECTED_IN_USE
0x898F	SOME_FILES_AFFECTED_READ_ONLY
0x8990	NO_FILES_AFFECTED_READ_ONLY
0x8993	NO_READ_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899E	INVALID_FILENAME
0x89A1	DIRECTORY_IO_ERROR

0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER
0x89FF	NO_FILES_FOUND_ERROR
0x89FF	File Exists Error

Remarks

The file resulting from **NWAFPCreateFile** is not opened; it is created as a normal Read/Write file with the system and hidden bits cleared.

For *AFPPathString*, byte 0 must be the length of the file name. The file name begins at byte 1 of the string. (Only include the file name---not the full path name---when calling **NWAFPCreateFile**.)

NCP Calls

0x2222 35 14 AFP 2.0 Create File

See Also

NWOpenNSEntry, **NWOpenDataStream**, **NWAFPDelete**, **NWAFPRename**

NWAFPDelete

Deletes an AFP file or directory
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPDelete (
   NWCONN_HANDLE   conn
   nuint16         volNum,
   nuint32         AFPEntryID,
   pnstr8          AFPPathString);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPDelete
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   AFPEntryID : nuint32;
   AFPPathString : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number of the directory entry location.

AFPEntryID

(IN) Specifies the AFP base ID.

AFPPathString

(IN) Points to the AFP directory path relative to *AFPEntryID*.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8805	NET_RECV_ERROR
0x8983	IO_ERROR_NETWORK_DISK
0x8988	INVALID_FILE_HANDLE
0x898A	NO_DELETE_PRIVILEGES
0x898D	SOME_FILES_AFFECTED_IN_USE
0x898E	NO_FILES_AFFECTED_IN_USE
0x898F	SOME_FILES_AFFECTED_READ_ONLY
0x8990	NO_FILES_AFFECTED_READ_ONLY
0x8993	NO_READ_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	Bad AFP Entry ID
0x899C	INVALID_PATH
0x899E	INVALID_FILENAME
0x899F	DIRECTORY_ACTIVE
0x89A0	DIRECTORY_NOT_EMPTY
0x89A1	DIRECTORY_IO_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER
0x89FF	NO_FILES_FOUND_ERROR
0x89FF	File Exists Error

Remarks

The directories to be deleted must be empty. Files to be deleted must be closed by all users.

For *AFPPathString*, byte 0 must be the length of the file name. The file name begins at byte 1 of the string. Include only the file name---not the full path name---when calling **NWAFPDelete**.

NCP Calls

0x2222 35 03 AFP Delete

See Also

NWAFPCreateDirectory, **NWAFPCreateFile**, **NWAFPGetEntryIDFromName**, **NWAFPGetEntryIDFromHandle**, **NWAFPGetEntryIDFromPathName**

NWAFPDirectoryEntry

Tests a directory entry to see if it is an AFP file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPDirectoryEntry (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPDirectoryEntry
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle for the path name.

path

(IN) Points to the path relative to *dirHandle*.

Return Values

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

0x0000	DOS File
0x0001	Macintosh File
0x8801	INVALID_CONNECTION
0x8983	IO_ERROR_NETWORK_DISK
0x8988	INVALID_FILE_HANDLE
0x8993	NO_READ_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x899C	INVALID_PATH
0x89A2	READ_FILE_WITH_RECORD_LOCKED

Remarks

The *dirHandle* and *path* parameters must be given in the DOS name space format.

NCP Calls

0x2222 22 5 Get Volume Number
0x2222 22 21 Get Volume Info With Handle
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 35 12 AFP Get Entry ID From Path Name
0x2222 35 15 AFP 2.0 Get File
0x2222 87 06 Obtain File or Subdirectory Information
0x2222 104 1 Ping for NDS NCP

See Also

NWAFPGetEntryIDFromPathName, NWGetVolumeInfoWithHandle, NWParsePath, NWGetVolumeNumber, NWAFPGetFileInformation, NWGetOwningNameSpace

NWAFPGetEntryIDFromHandle

Returns an AFP entry ID for the specified NetWare handle
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPGetEntryIDFromHandle (
   NWCONN_HANDLE   conn,
   pnuint8         NWHandle,
   pnuint16        volNum,
   pnuint32        AFPEntryID,
   pnuint8         forkIndicator);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPGetEntryIDFromHandle
  (conn : NWCONN_HANDLE;
   NWHandle : pnuint8;
   volNum : pnuint16;
   AFPEntryID : pnuint32;
   forkIndicator : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

NWHandle

(IN) Points to the 6-byte NetWare handle for the path name.

volNum

(OUT) Points to the volume number of the directory entry location.

AFPEntryID

(OUT) Points to the AFP file entry ID.

forkIndicator

forkIndicator

(OUT) Points to the fork indicator (0 = data; 1 = resource).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE
0x8998	VOLUME_DOES_NOT_EXIST
0x899C	INVALID_PATH

Remarks

AFPEntryID points to the AFP file ID. It is not the AFP base ID. INVALID_PATH will be returned if you use the *AFPEntryID* as the AFP base ID.

NCP Calls

0x2222 35 06 AFP Get Entry ID From NetWare Handle

See Also

NWAFPGetEntryIDFromName, **NWAFPGetEntryIDFromPathName**, **NWAFPGetFileInformation**, **NWAFPAllocTemporaryDirHandle**

NWAFPGetEntryIDFromName

Returns a unique AFP entry ID from an AFP entry ID of a parent and a modifying path

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPGetEntryIDFromName (
   NWCONN_HANDLE   conn,
   nuint16         volNum,
   nuint32         AFPEntryID,
   pnstr8          AFPPathString,
   pnuint32        newAFPEntryID);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPGetEntryIDFromName
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   AFPEntryID : nuint32;
   AFPPathString : pnstr8;
   newAFPEntryID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number of the directory entry location.

AFPEntryID

(IN) Specifies the unique AFP base ID.

AFPPathString

(IN) Points to the path string modifying *AFPEntryID*.

newAFPEntryID

(OUT) Points to the AFP entry ID of the given path.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE
0x8983	IO_ERROR_NETWORK_DISK
0x8993	NO_READ_RRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x899C	INVALID_PATH
0x8998	VOLUME_DOES_NOT_EXIST
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER
0x89FF	NO_FILES_FOUND_ERROR

NCP Calls

0x2222 35 04 AFP Get Entry ID From Name

See Also

NWAFPGetEntryIDFromHandle, **NWAFPGetEntryIDFromPathName**

NWAFPGetEntryIDFromPathName

Returns a unique 32-bit AFP file or directory ID, given a combination of path and directory handle

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPGetEntryIDFromPathName (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   pnuint32        AFPEntryID);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPGetEntryIDFromPathName
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   AFPEntryID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle for *path*.

path

(IN) Points to the path given relative to the directory handle.

AFPEntryID

(OUT) Points to the AFP base ID.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	IO_ERROR_NETWORK_DISK
0x8988	INVALID_FILE_HANDLE
0x8993	NO_READ_RRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER
0x89FF	NO_FILES_FOUND_ERROR

Remarks

The directory base and path specifications must be given in DOS name space format.

NCP Calls

0x2222 35 12 AFP Get Entry ID From Path Name

See Also

NWAFPGetEntryIDFromHandle, **NWAFPGetEntryIDFromName**, **NWAFPGetFileInformation**, **NWAFPAllocTemporaryDirHandle**, **NWAllocTempNSDirHandle2**

NWAFPGetFileInformation

Returns AFP information for a directory or file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPGetFileInformation (
   NWCONN_HANDLE              conn,
   nuint16                    volNum,
   nuint32                    AFPEntryID,
   nuint16                    reqMask,
   pnstr8                     AFPPathString,
   nuint16                    structSize,
   NW_AFP_FILE_INFO N_FAR  *AFPFileInfo);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPGetFileInformation
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   AFPEntryID : nuint32;
   reqMask : nuint16;
   AFPPathString : pnstr8;
   structSize : nuint16;
   Var AFPFileInfo : NW_AFP_FILE_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number of the directory entry location.

AFPEntryID

(IN) Specifies the unique AFP base ID.

reqMask

(IN) Specifies the request bit mask information.

AFPPathString

(IN) Points to the AFP directory path relative to *AFPEntryID*.

structSize

(IN) Specifies the request AFPFILEINFO buffer size.

AFPFileInfo

(OUT) Points to AFPFILEINFO returning AFP file information.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE
0x8983	IO_ERROR_NETWORK_DISK
0x8993	NO_READ_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899C	INVALID_PATH
0x89A1	DIRECTORY_I/O_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER
0x89FF	Failure. NO_FILES_FOUND_ERROR

Remarks

Valid bit map information request values follow for *reqMask*: (Bits can be ORed together.)

C Values	Pascal Values	Value Names
0x0001	$0001	AFP_GET_ATTRIBUTES
0x0002	$0002	AFP_GET_PARENT_ID
0x0004	$0004	AFP_GET_CREATE_DATE
0x0008	$0008	AFP_GET_ACCESS_DATE

0x0010	$0010	AFP_GET_MODIFY_DATETIME
0x0020	$0020	AFP_GET_BACKUP_DATETIME
0x0040	$0040	AFP_GET_FINDER_INFO
0x0080	$0080	AFP_GET_LONG_NAME
0x0100	$0100	AFP_GET_ENTRY_ID
0x0200	$0200	AFP_GET_DATA_LEN
0x0400	$0400	AFP_GET_RESOURCE_LEN
0x0800	$0800	AFP_GET_NUM_OFFSPRING
0x1000	$1000	AFP_GET_OWNER_ID
0x2000	$2000	AFP_GET_SHORT_NAME
0x4000	$4000	AFP_GET_ACCESS_RIGHTS
0x8000	$8000	AFP_GET_PRO_DOS_INFO
0xffff	$ffff	AFP_GET_ALL

NCP Calls

0x2222 35 15 AFP 2.0 Get File

See Also

NWAFPSetFileInformation, NWAFPScanFileInformation

NWAFPOpenFileFork

Opens an AFP file fork from a DOS environment
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPOpenFileFork (
   NWCONN_HANDLE          conn,
   nuint16                volNum,
   nuint32                AFPEntryID,
   nuint8                 forkIndicator,
   nuint8                 accessMode,
   pnstr8                 AFPPathString,
   pnuint32               fileID,
   pnuint32               forkLength,
   pnuint8                NWHandle,
   NWFILE_HANDLE N_FAR  *DOSFileHandle);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPOpenFileFork
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   AFPEntryID : nuint32;
   forkIndicator : nuint8;
   accessMode : nuint8;
   AFPPathString : pnstr8;
   fileID : pnuint32;
   forkLength : pnuint32;
   NWHandle : pnuint8;
   Var DOSFileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number of the directory entry location.

AFPEntryID

(IN) Specifies the AFP base ID.

forkIndicator

(IN) Specifies the data or resource fork indicator (0=data; 1=resource).

accessMode

(IN) Specifies the file access mode indicator. (AR_READ and/or AR_WRITE should be set.)

AFPPathString

(IN) Points to the AFP directory path relative to *AFPEntryID*.

fileID

(OUT) Points to the file entry ID.

forkLength

(OUT) Points to the length of the opened fork.

NWHandle

(OUT) Points to the 6-byte NetWare file handle.

DOSFileHandle

(OUT) Points to the file handle.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8980	FILE_IN_USE_ERROR
0x8981	NO_MORE_FILE_HANDLES
0x8988	INVALID_FILE_HANDLE
0x8983	IO_ERROR_NETWORK_DISK
0x8993	NO_READ_PRIVILEGES
0x8994	NO_WRITE_PRIVILEGES_OR_READONLY
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899C	Invalid AFP Path String

0x89A1	DIRECTORY_IO_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER
0x89FF	LOCK_ERROR, NO_FILES_FOUND_ERROR

Remarks

If a file does not exist, **NWAFPOpenFileFork** returns NO_FILES_FOUND_ERROR (0x89FF).

If an existing file does not have a resource or data file fork associated with it, **NWAFPOpenFileFork** will automatically create and open the specified file fork.

These constants are used by **NWAFPOpenFileFork** to identify access rights attributes.

C Value	Pascal Value	Value Name
0x0001	$0001	AR_READ
0x0002	$0002	AR_WRITE
0x0001	$0001	AR_READ_ONLY
0x0002	$0002	AR_WRITE_ONLY
0x0004	$0004	AR_DENY_READ
0x0008	$0008	AR_DENY_WRITE
0x0010	$0010	AR_COMPATIBILITY
0x0040	$0040	AR_WRITE_THROUGH
0x0100	$0100	AR_OPEN_COMPRESSED

NCP Calls

0x2222 35 08 AFP Open File Fork

See Also

NWAFPCreateFile, **NWAFPGetFileInformation**, **NWAFPGetEntryIDFromName**

NWAFPRename

Renames an AFP file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPRename (
   NWCONN_HANDLE   conn,
   nuint16         volNum,
   nuint32         AFPSourceEntryID,
   nuint32         AFPDestEntryID,
   pnstr8          AFPSrcPath,
   pnstr8          AFPDstPath);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPRename
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   AFPSourceEntryID : nuint32;
   AFPDestEntryID : nuint32;
   AFPSrcPath : pnstr8;
   AFPDstPath : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number of the directory entry location.

AFPSourceEntryID

(IN) Specifies the AFP source base ID.

AFPDestEntryID

(IN) Specifies the AFP destination base ID.

AFPSrcPath

(IN) Points to the AFP source directory path relative to *AFPSourceEntryID*.

AFPDstPath

(IN) Points to the AFP destination directory path, relative to *AFPDestEntryID*.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8984	NO_CREATE_PRIVILEGES
0x8988	INVALID_FILE_HANDLE
0x8983	IO_ERROR_NETWORK_DISK
0x898B	NO_RENAME_PRIVILEGES
0x898E	NO_FILES_AFFECTED_IN_USE
0x8990	NO_FILES_AFFECTED_READ_ONLY
0x8992	NO_FILES_RENAMED_NAME_EXISTS
0x8993	NO_READ_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899C	Invalid AFP Path String
0x899E	INVALID_FILENAME
0x89A1	DIRECTORY_IO_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER
0x89FF	NO_FILES_FOUND_ERROR

NCP Calls

0x2222 35 07 AFP Rename

See Also

File Service Group

NWNSRename, NWAFPGetEntryIDFromName

NWAFPScanFileInformation

Scans a directory and returns AFP file/directory information
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPScanFileInformation (
   NWCONN_HANDLE              conn,
   nuint16                    volNum,
   nuint32                    AFPEntryID,
   pnuint32                   AFPLastSeenID,
   nuint16                    searchMask,
   nuint16                    reqMask,
   pnstr8                     AFPPathString,
   nuint16                    structSize,
   NW_AFP_FILE_INFO N_FAR  *AFPFileInfo);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPScanFileInformation
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   AFPEntryID : nuint32;
   AFPLastSeenID : pnuint32;
   searchMask : nuint16;
   reqMask : nuint16;
   AFPPathString : pnstr8;
   structSize : nuint16;
   Var AFPFileInfo : NW_AFP_FILE_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number of the directory entry location.

AFPEntryID

(IN) Specifies the AFP base ID.

AFPLastSeenID

(IN) Points to *AFPEntryID*.

searchMask

(IN) Specifies the search mask.

reqMask

(IN) Specifies the request bit mask information.

AFPPathString

(IN) Points to the AFP directory path relative to *AFPEntryID*.

structSize

(IN) Specifies the size of the AFPFILEINFO buffer.

AFPFileInfo

(OUT) Points to AFPFILEINFO returning AFP file information.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	IO_ERROR_NETWORK_DISK
0x8988	INVALID_PATH
0x8993	NO_READ_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER
0x89FF	NO_FILES_FOUND_ERROR

Remarks

AFPLastSeenID should be initialized to -1 on the first iteration.

Valid bit map information request values follow for *reqMask*. (Bits can be ORed together.)

C Values	Pascal Values	Value Names
0x0001	$0001	AFP_GET_ATTRIBUTES
0x0002	$0002	AFP_GET_PARENT_ID
0x0004	$0004	AFP_GET_CREATE_DATE
0x0008	$0008	AFP_GET_ACCESS_DATE
0x0010	$0010	AFP_GET_MODIFY_DATETIME
0x0020	$0020	AFP_GET_BACKUP_DATETIME
0x0040	$0040	AFP_GET_FINDER_INFO
0x0080	$0080	AFP_GET_LONG_NAME
0x0100	$0100	AFP_GET_ENTRY_ID
0x0200	$0200	AFP_GET_DATA_LEN
0x0400	$0400	AFP_GET_RESOURCE_LEN
0x0800	$0800	AFP_GET_NUM_OFFSPRING
0x1000	$1000	AFP_GET_OWNER_ID
0x2000	$2000	AFP_GET_SHORT_NAME
0x4000	$4000	AFP_GET_ACCESS_RIGHTS
0x8000	$8000	AFP_GET_PRO_DOS_INFO
0xffff	$ffff	AFP_GET_ALL

NWSEARCH_MASK is defined as follows:

C Values	Pascal Values	Value Names
0x0000	$0000	AFP_SA_NORMAL
0x0100	$0100	AFP_SA_HIDDEN
0x0200	$0200	AFP_SA_SYSTEM
0x0400	$0400	AFP_SA_SUBDIR
0x0800	$0800	AFP_SA_FILES
0xF00	$0F00	AFP_SA_ALL

AFPFILEINFO *attributes* follow:

```
0x0001 = Search Mode
0x0002 = Search Mode
```

```
0x0004 = Search Mode
0x0008 = Undefined
0x0010 = Transaction
0x0020 = Index
0x0040 = Read Audit
0x0080 = Write Audit
0x0100 = Read Only
0x0200 = Hidden
0x0400 = System
0x0800 = Execute Only
0x1000 = Subdirectory
0x2000 = Archive
0x4000 = Undefined
0x8000 = Shareable File
```

NCP Calls

0x2222 35 17 AFP 2.0 Scan File Information

See Also

NWAFPGetFileInformation

NWAFPSetFileInformation

Sets AFP information for a file or directory
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPSetFileInformation (
   NWCONN_HANDLE            conn,
   nuint16                  volNum,
   nuint32                  AFPBaseID,
   nuint16                  reqMask,
   pnstr8                   AFPPathString,
   nuint16                  structSize,
   NW_AFP_SET_INFO N_FAR  *AFPSetInfo);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPSetFileInformation
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   AFPBaseID : nuint32;
   reqMask : nuint16;
   AFPPathString : pnstr8;
   structSize : nuint16;
   Var AFPSetInfo : NW_AFP_SET_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number of the directory entry location.

AFPBaseID

(IN) Specifies the AFP base ID.

reqMask

(IN) Specifies the request bit mask information.

AFPPathString

(IN) Points to the AFP directory path relative to *AFPBaseID*.

structSize

(IN) Specifies the size of the AFPSETINFO buffer.

AFPSetInfo

(IN) Points to AFPSETINFO to set AFP file information.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8901	ERR_INSUFFICIENT_SPACE
0x8988	INVALID_FILE_HANDLE
0x8901	ERR_INSUFFICIENT_SPACE
0x8983	IO_ERROR_NETWORK_DISK
0x8993	NO_READ_PRIVILEGES
0x8994	NO_WRITE_PRIVILEGES_OR_READONLY
0x8995	FILE_DETACHED
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FD	BAD_STATION_NUMBER
0x89FF	Failure; NO_FILES_FOUND_ERROR

Remarks

The following constants are used by **NWAFPSetFileInformation** to manipulate *requestMask*. They are also used in by *NWSEARCH_MASK* in **NWAFPScanFileInformation**.

C	Pascal Values	Value Names

Values		
0x0001	$0001	AFP_GET_ATTRIBUTES
0x0002	$0002	AFP_GET_PARENT_ID
0x0004	$0004	AFP_GET_CREATE_DATE
0x0008	$0008	AFP_GET_ACCESS_DATE
0x0010	$0010	AFP_GET_MODIFY_DATETIME
0x0020	$0020	AFP_GET_BACKUP_DATETIME
0x0040	$0040	AFP_GET_FINDER_INFO
0x0080	$0080	AFP_GET_LONG_NAME
0x0100	$0100	AFP_GET_ENTRY_ID
0x0200	$0200	AFP_GET_DATA_LEN
0x0400	$0400	AFP_GET_RESOURCE_LEN
0x0800	$0800	AFP_GET_NUM_OFFSPRING
0x1000	$1000	AFP_GET_OWNER_ID
0x2000	$2000	AFP_GET_SHORT_NAME
0x4000	$4000	AFP_GET_ACCESS_RIGHTS
0x8000	$8000	AFP_GET_PRO_DOS_INFO
0xffff	$ffff	AFP_GET_ALL

These constants identify AFP entries to be included in **NWAFPSetFileInformation**.

C Values	Pascal Values	Value Names
0x0000	$0000	AFP_SA_NORMAL
0x0100	$0100	AFP_SA_HIDDEN
0x0200	$0200	AFP_SA_SYSTEM
0x0400	$0400	AFP_SA_SUBDIR
0x0800	$0800	AFP_SA_FILES
0xF00	$0F00	AFP_SA_ALL

Valid bit map information request values follow for *reqMask*: (Bits can be ORed together.)

C Values	Pascal Values	Value Names
0x0001	$0001	AFP_SET_ATTRIBUTES

0x0004	$0004	AFP_SET_CREATE_DATE
0x0008	$0008	AFP_SET_ACCESS_DATE
0x0010	$0010	AFP_SET_MODIFY_DATETIME
0x0020	$0020	AFP_SET_BACKUP_DATETIME
0x0040	$0040	AFP_SET_FINDER_INFO
0x8000	$8000	AFP_SET_PRO_DOS_INFO

AFPSETINFO *attributes* follow:

```
0x0001 = Search Mode
0x0002 = Search Mode
0x0004 = Search Mode
0x0008 = Undefined
0x0010 = Transaction
0x0020 = Index
0x0040 = Read Audit
0x0080 = Write Audit
0x0100 = Read Only
0x0200 = Hidden
0x0400 = System
0x0800 = Execute Only
0x1000 = Subdirectory
0x2000 = Archive
0x4000 = Undefined
0x8000 = Shareable File
```

NCP Calls

0x2222 35 16 AFP 2.0 Set File Information

See Also

NWAFPGetFileInformation, **NWAFPScanFileInformation**, **NWSetLongName**

NWAFPSupported

Reports whether the AFP is supported on a server volume
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: AFP

Syntax

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

NWCCODE NWAPI NWAFPSupported (
   NWCONN_HANDLE   conn,
   nuint16         volNum);
```

Pascal Syntax

```
#include <nwafp.inc>

Function NWAFPSupported
  (conn : NWCONN_HANDLE;
   volNum : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number to test.

Return Values

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

0x0000	AFP supported
Non-zero	AFP not supported

NCP Calls

0x2222 22 6 Get Volume Name
0x2222 35 12 AFP Get Entry ID From Path Name

See Also

NWReadNSInfo, **NWGetVolumeName**, **NWAFPGetEntryIDFromPathName**

AFP: Structures

NW_AFP_FILE_INFO

Defines file information for AFP files
Service: AFP
Defined In: nwafp.h

Structure

```
typedef struct
{
   nuint32   entryID;
   nuint32   parentID;
   nuint16   attributes;
   nuint32   dataForkLength;
   nuint32   resourceForkLength;
   nuint16   numOffspring;
   nuint16   creationDate;
   nuint16   accessDate;
   nuint16   modifyDate;
   nuint16   modifyTime;
   nuint16   backupDate;
   nuint16   backupTime;
   nuint8    finderInfo[32];
   nstr8     longName[34];
   nuint32   ownerID;
   nstr8     shortName[14];
   nuint16   accessPrivileges;
   nuint8    proDOSInfo[6];
} NW_AFP_FILE_INFO, AFPFILEINFO;
```

Pascal Structure

```
Defined in nwafp.inc

AFPFILEINFO = Record
   entryID : nuint32;
   parentID : nuint32;
   attributes : nuint16;
   dataForkLength : nuint32;
   resourceForkLength : nuint32;
   numOffspring : nuint16;
   creationDate : nuint16;
   accessDate : nuint16;
   modifyDate : nuint16;
   modifyTime : nuint16;
   backupDate : nuint16;
   backupTime : nuint16;
   finderInfo : Array[0..31] Of nuint8;
   longName : Array[0..33] Of nstr8;
```

```
    ownerID : nuint32;
    shortName : Array[0..13] Of nstr8;
    accessPrivileges : nuint16;
    proDOSInfo : Array[0..5] Of nuint8
End;
```

Fields

entryID

Indicates the unique AFP identifier of a file or directory.

parentID

Indicates the unique AFP identifier for the parent directory. The root will be 0.

attributes

Indicates the set of bits identifying the entry's attributes:

0x0001 = Search Mode
0x0002 = Search Mode
0x0004 = Search Mode
0x0008 = Undefined
0x0010 = Transaction
0x0020 = Index
0x0040 = Read Audit
0x0080 = Write Audit
0x0100 = Read Only
0x0200 = Hidden
0x0400 = System
0x0800 = Execute Only
0x1000 = Subdirectory
0x2000 = Archive
0x4000 = Undefined
0x8000 = Shareable File

dataForkLength

Indicates the data size of the target AFP file. If *pathModString* specifies an AFP directory, *dataForkLength* returns a zero (0).

resourceForkLength

Indicates the resource fork size of the target AFP file. If *pathModString* specifies an AFP directory, *resourceForkLength* returns a zero.

numOffspring

Indicates the number of files and subdirectiries contained within the specified directory. If the AFP directory or file path specifies an AFP file, *numOffspring* returns a zero (0).

creationDate

Indicates the creation date (in AFP format) of the target directory or

file.

accessDate

Indicates when the target AFP file was last accessed (returned in AFP format). If *pathModString* specifies an AFP directory, *accessDate* returns a zero.

modifyDate

Indicates the last modified date (in AFP format) of the target AFP file. If *pathModString* specifies an AFP directory, *modifyDate* returns zero.

modifyTime

Indicates the last modified time (in AFP format) of the target AFP file. If *pathModString* specifies an AFP directory, *modifyTime* returns zero.

backupDate

Indicates the last backup date (in AFP format) of the specified directory or file.

backupTime

Indicates the last backup time (in AFP format) of the specified directory or file.

finderInfo

Indicates the 32-byte finder information structure associated with each AFP directory or file.

longName

Indicates the AFP directory or file name of the specified directory or file. An AFP directory or filename can be from 1 to 31 characters long. *longName* is a null-terminated ASCII string. One extra byte has been added for the NULL terminator and another byte has been added to ensure word alignment.

ownerID

Indicates the 4-byte bindery object ID of the object creating or last modifying the file.

shortName

Indicates the NetWare® directory or file name of the specified directory or file in the DOS name space. A NetWare directory or file name is in DOS 8.3 format. *shortName* is a null-terminated ASCII string. One extra byte has been added for the NULL terminator and another byte has been added to ensure word alignment.

accessPrivileges

Indicates the one-word bit mask of the calling station's privileges for accessing the specified file or directory.

proDOSInfo

Indicates the 6-byte structure defined in Apple documentation.

NW_AFP_SET_INFO

Defines Apple file attributes
Service: AFP
Defined In: nwafp.h

Structure

```
typedef struct
{
   nuint16   attributes;
   nuint16   creationDate;
   nuint16   accessDate;
   nuint16   modifyDate;
   nuint16   modifyTime;
   nuint16   backupDate;
   nuint16   backupTime;
   nuint8    finderInfo[32];
   nuint8    proDOSInfo[6];
} NW_AFP_SET_INFO, AFPSETINFO;
```

Pascal Structure

```
Defined in nwafp.inc

AFPSETINFO = Record
   attributes : nuint16;
   creationDate : nuint16;
   accessDate : nuint16;
   modifyDate : nuint16;
   modifyTime : nuint16;
   backupDate : nuint16;
   backupTime : nuint16;
   finderInfo : Array[0..31] Of nuint8;
   proDOSInfo : Array[0..5] Of nuint8
 End;
```

Fields

attributes

Indicates the file attributes.

creationDate

Indicates the creation date (in AFP format) of the target directory or file.

accessDate

Indicates when the target AFP file was last accessed (returned in AFP

format).

modifyDate

Indicates the last modified date (in AFP format) of the target AFP file.

modifyTime

Indicates the last modified time (in AFP format) of the target AFP file.

backupDate

Indicates the last date (in AFP format) the file was backed up.

backupTime

Indicates the time (in AFP format) the file was last backed up.

finderInfo

Indicates the information defined in Apple documentation.

proDOSInfo

Indicates the 6-byte structure defined in Apple documentation.

RECPKT_AFPFILEINFO

Is the structure actually returned from the NCP call
Service: AFP
Defined In: nwafp.h

Structure

```
typedef struct
{
   nuint32   entryID;
   nuint32   parentID;
   nuint16   attributes;
   nuint32   dataForkLength;
   nuint32   resourceForkLength;
   nuint16   numOffspring;
   nuint16   creationDate;
   nuint16   accessDate;
   nuint16   modifyDate;
   nuint16   modifyTime;
   nuint16   backupDate;
   nuint16   backupTime;
   nuint8    finderInfo[32];
   nstr8     longName[32];
   nuint32   ownerID;
   nstr8     shortName[12];
   nuint16   accessPrivileges;
   nuint8    proDOSInfo[6];
} RECPKT_AFPFILEINFO;
```

Pascal Structure

```
Defined in nwafp.inc

RECPKT_AFPFILEINFO = Record
   entryID : nuint32;
   parentID : nuint32;
   attributes : nuint16;
   dataForkLength : nuint32;
   resourceForkLength : nuint32;
   numOffspring : nuint16;
   creationDate : nuint16;
   accessDate : nuint16;
   modifyDate : nuint16;
   modifyTime : nuint16;
   backupDate : nuint16;
   backupTime : nuint16;
   finderInfo : Array[0..31] Of nuint8;
   longName : Array[0..31] Of nstr8;
```

```
    ownerID : nuint32;
    shortName : Array[0..11] Of nstr8;
    accessPrivileges : nuint16;
    proDOSInfo : Array[0..5] Of nuint8
End;
```

Fields

entryID

Indicates the unique AFP identifier of a file or directory.

parentID

Indicates the unique AFP identifier for the parent directory. The root will be 0.

attributes

Indicates the set of bits identifying the entry's attributes:

0x0001 = Search Mode
0x0002 = Search Mode
0x0004 = Search Mode
0x0008 = Undefined
0x0010 = Transaction
0x0020 = Index
0x0040 = Read Audit
0x0080 = Write Audit
0x0100 = Read Only
0x0200 = Hidden
0x0400 = System
0x0800 = Execute Only
0x1000 = Subdirectory
0x2000 = Archive
0x4000 = Undefined
0x8000 = Shareable File

dataForkLength

Indicates the data size of the target AFP file. If *pathModString* specifies an AFP directory, *dataForkLength* returns a zero (0).

resourceForkLength

Indicates the resource fork size of the target AFP file. If *pathModString* specifies an AFP directory, *resourceForkLength* returns a zero.

numOffspring

Indicates the number of files and subdirectiries contained within the specified directory. If the AFP directory or file path specifies an AFP file, *numOffspring* returns a zero (0).

creationDate

Indicates the creation date (in AFP format) of the target directory or

file.

accessDate

Indicates when the target AFP file was last accessed (returned in AFP format). If *pathModString* specifies an AFP directory, *accessDate* returns a zero.

modifyDate

Indicates the last modified date (in AFP format) of the target AFP file. If *pathModString* specifies an AFP directory, *modifyDate* returns zero.

modifyTime

Indicates the last modified time (in AFP format) of the target AFP file. If *pathModString* specifies an AFP directory, *modifyTime* returns zero.

backupDate

Indicates the last backup date (in AFP format) of the specified directory or file.

backupTime

Indicates the last backup time (in AFP format) of the specified directory or file.

finderInfo

Indicates the 32-byte finder information structure associated with each AFP directory or file.

longName

Indicates the AFP directory or file name of the specified directory or file. An AFP directory or filename can be from 1 to 31 characters long.

ownerID

Indicates the 4-byte bindery object ID of the object creating or last modifying the file.

shortName

Indicates the NetWare® directory or file name of the specified directory or file in the DOS name space. A NetWare directory or file name is in DOS 8.3 format.

accessPrivileges

Indicates the one-word bit mask of the calling station's privileges for accessing the specified file or directory.

proDOSInfo

Indicates the 6-byte structure defined in Apple documentation.

Data Migration

Data Migration: Guides

Data Migration: Concepts

Data Migration Introduction

Data Migration Volume Information

Support Module Information

Data Migration Functions

Migrating Files Example

Additional Links

Data Migration: Functions

Data Migration: Structures

Parent Topic:

File Overview

Data Migration: Examples

Migrating Files Example

The following is an example of migrating files on a monthly basis.

> **NOTE:** This example contains DOS specific screen functions that should be ignored on non-DOS platforms.

Migrating Files on a Monthly Basis

```
# define N_PLAT_DOS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dir.h>
#include <ntypes.h>
#include <dos.h>
#include <nwmigrate.h>
#include <nwmisc.h>
#include <nwdpath.h>
#include <nwnamespc.h>

extern unsigned int _stklen = 8192;

#define HCSSSMID 0x640F0F6DL

DWORD HCSSSupportModuleID = HCSSSMID;

main(int argc, char *argv[])
{
   struct
   {
      WORD      len;
      BYTE      subFunction;
      nuint32   supportModuleID;
      DWORD     vol;
      DWORD     TargetDirectoryBase;
   } reqBuf;

   struct Info3RepStruct
   {
      DWORD BlockSizeInSectors;
      DWORD TotalBlocks;
      DWORD UsedBlocks;
```

```
} info3Rep;

struct Info0RepStruct
{
   DWORD rIOStatus;
   DWORD rInfoBlockSize;
   DWORD rAvailSpace;
   DWORD rUsedSpace;
   BYTE  rSMString[128];
   DWORD majorVersion;
   DWORD minorVersion;
   DWORD revision;
   DWORD year;
   DWORD month;
   DWORD day;

} info0Rep;

NWDIR_HANDLE    dirHandle;
NWCONN_HANDLE   conn;
NW_FRAGMENT     reqFrag[1], repFrag[1];
NW_IDX          idxStruct;
DWORD           retLen;
nuint16         drive;
struct          ffblk ffblk;
NWCCODE         ccode;
nstr8           path[MAXPATH], dirname[MAXPATH], dirPath[MAXPATH],
                   *months[] = {"January", "February", "March",
                   "April", "May", "June", "July", "August",
                   "September", "October", "November", "December"};

if (argc !=2)
{
   printf("Usage: migrate <filename>\n");
   exit(1);
}
strupr( argv[1]);

getcwd(dirname, MAXPATH);
drive = dirname[0]-64;

ccode = NWCallsInit(NULL, NULL):
if(ccode)
   exit(1);

ccode = findfirst(argv[1], &ffblk, 0);
if(ccode)
   exit(1);

ccode = NWGetDriveInformation(drive, 0, &conn, &dirHandle, 0,
                              dirPath);
if(ccode)
```

```
   exit(1);

ccode = NWGetDirectoryBase(conn, dirHandle, dirPath, NW_NS_DOS,
                           &idxStruct);
   if(ccode)
      exit(1);

   ccode = NWSetDefaultSupportModule(conn, &HCSSSupportModuleID);
   if(ccode)
      exit(1);
   {
      sprintf(path, "%s\\%s", dirPath, argv[1]); /* path minus
              drive */
    /* Get media capacities relative to the path. */

      reqBuf.len                 = 9;
      reqBuf.subFunction         = 135; /* For SM info level 3 */
      reqBuf.supportModuleID     = HCSSMID;
      reqBuf.vol                 = idxStruct.volNumber;
      reqBuf.TargetDirectoryBase = idxStruct.dstDirBase;

      reqFrag[0].fragAddress = &reqBuf'
      reqFrag[0].fragSize    = sizeof(reqBuf);

      repFrag[0].fragAddress = &info3Rep;
      repFrag[0].fragSize    = sizeof(struct Info3RepStruct);

      ccode = NWRequest(conn, 90, 1, reqFrag, 1, repFrag);
      if(ccode)
         exit(1);
         {

   /* Make sure the file is not greater than the available space on
      media. */

          if(((ffblk.ff_fsize+511)/512) <
             ((info3Rep.TotalBlocks - info3Rep.UsedBlocks)
              * info3Rep.BlockSizeInSectors))

           {
           /* Migrate the file. */
              switch(ccode = NWMoveFileToDM(conn, dirHandle,
                    path, NW_NS_DOS, HCSSSupportModuleID, 1))

           {
               case 0:
   /* Get media capacities relative to the path. */

               retLen = sizeof(struct
                        Info0RepStruct);

               ccode = NWGetSupportModuleInfo(
```

```
                                    conn, 0L,HCSSSupportModuleID,
                                    (BYTE far*)&info0Rep.rIOStatus,
                                    &retLen);

                                if(ccode)
                                   printf("\nNWGetSupport
                                          ModuleInfo() Failure
                                          0x%x\n",);
                                   else
                                   {
                                      printf("Info block size:
                                             %lu\n", info0Rep.rInfo
                                             BlockSize);

                                      printf("Total available space
                                             (in sectors): %lu\n",
                                             info0Rep.rAvailSpace);

                                      printf("Total used space (in
                                             sectors): %lu\n, info0Rep.
                                             rUsedSpace);

                                      printf("Support module name:
                                             %s\n", info0Rep.rSMString);

                                      printf("Support module version:
                                             %lu.%lu rev. %lu\n",
                                             info0Rep.majorVersion,
                                             info0Rep.minorVersion,
                                             info0Rep.revision);

                                      printf("Support module date:
                                              %s %lu, %lu\n", months
                                              [(int)(info0Rep.month-1)],
                                              info0Rep.day, info0Rep.year);

                                   }
                                   break;

                                case 0x8995:
                                   printf("File is already migrated.\n");

                                   break;

                                default:
                                   printf("Unsuccessful completion code:
                                           x0%x\n", ccode);
                             }
                          }
                       }
                    }
                 return(0);
```

```
}
```

Parent Topic:

Data Migration: Guides

Data Migration: Concepts

Data Migration Functions

These functions move files to and from remote storage, return data migration information for files and volumes, and return information about the Data Migrator and support modules.

Function	Comment
NWMoveFileToDM	Moves a file's data to an online, long term storage media but leaves the file visible on the NetWare® volume.
NWMoveFileFromDM	Moves a file's data from an online, long term storage media to a NetWare volume.
NWGetDataMigratorInfo	Returns version numbers for the Data Migrator NLM. Use this function to test whether the Data Migrator is loaded.
NWGetDefaultSupportModule	Returns the default support module for reading and writing migrated data.
NWGetDMFileInfo	Returns information about migrated files.
NWGetDMVolumeInfo	Returns information about the data that has been migrated in relation to the specified volume.
NWGetSupportModuleInfo	Can return either a list of data migration support module IDs or information about a specific support module.
NWSetDefaultSupportModule	Sets the default support module for reading and writing migrated data.

Parent Topic:

Data Migration: Guides

Data Migration Introduction

Data Migration Services enable client applications to move NetWare® files to supplementary nearline storage devices. Nearline storage devices include another volume, another server, another media type, another file system, a tape or even a jukebox. Migrated files are still readily accessible, although the files themselves are remote. When the files are accessed, they are de-migrated in real time to primary storage. The files remain in the file system's directory structure and all file information stays intact.

Retrieval time for migrated files varies, depending on the nearline storage device. Retrieval from a CD ROM or disk subsystem is nearly as fast as retrieval from a NetWare volume.

Files migrated are still accessed through the NetWare file system. For example, files migrated to a jukebox remain visible in the NetWare directory and when a user attempts to access one of these files, the system retrieves the data from the jukebox.

A Data Migrator NLM™ application administers data migration and is available from Novell®. Support module NLM applications register with the Data Migrator to provide access to specific storage schemas. The Novell Data Migrator can register up to 32 support modules.

Users and administrators determine the criteria for migrating files. These criteria typically specify seldom accessed files or files that require excessive storage space, such as large database files. Users can migrate an unlimited number of files.

Parent Topic:

Data Migration: Guides

Data Migration Volume Information

NWGetDMVolumeInfo returns information about the Data Migrator NLM on a volume. Data migration volume information includes:

Number of migrated files

Total size of migrated data

Size of data on the migration media

Amount of limbo space

Limbo space refers to migrated files that have been restored to the file system but not removed from remote storage. Generally, files are retained in remote storage after they have been migrated until the file is either deleted or re-migrated.

Parent Topic:

Data Migration: Guides

Support Module Information

All available support modules are registered with the Data Migrator under a support module ID. Call **NWGetSupportModuleInfo** to receive a list of support modules. After receiving the IDs, use the same function to receive information about individual support modules.

The support module list is returned as aSUPPORT_MODULE_IDS structure. It contains an array of support module IDs.

Information about individual modules is returned as a SUPPORT_MODULE_INFO structure.

IO status

Block size

Available space

Space in-use

Information specific to the module can also be returned as a length-preceded string.

Parent Topic:

Data Migration: Guides

Data Migration: Functions

NWGetDataMigratorInfo

Returns information about the data migrator
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Data Migration

Syntax

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

NWCCODE N_API NWGetDataMigratorInfo (
    NWCONN_HANDLE   conn,
    pnuint32        DMPresentFlag,
    pnuint32        majorVersion,
    pnuint32        minorVersion,
    pnuint32        DMSMRegistered);
```

Pascal Syntax

```
#include <nwmigrat.inc>

Function NWGetDataMigratorInfo
  (conn : NWCONN_HANDLE;
   DMPresentFlag : pnuint32;
   majorVersion : pnuint32;
   minorVersion : pnuint32;
   DMSMRegistered : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

DMPresentFlag

(OUT) Points to a flag. If equal to -1, the DM NLM has been loaded and is running; if equal to 0, the DM NLM is not loaded.

majorVersion

(OUT) Points to the data migrator major version number.

minorVersion

(OUT) Points to the data migrator minor version number.

DMSMRegistered

(OUT) Points to a flag indicating if the support module has been registered with the data migrator: non-zero = support module was registered, zero = support module was not registered.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x89FB	Data Migration is not supported

NCP Calls

0x2222 90 131 Migrator Status Info

See Also

NWGetDMVolumeInfo, NWGetDMFileInfo

NWGetDefaultSupportModule

Returns the default Read/Write Support Module ID for data migration
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Data Migration

Syntax

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

NWCCODE N_API NWGetDefaultSupportModule (
    NWCONN_HANDLE   conn,
    pnuint32        supportModuleID);
```

Pascal Syntax

```
#include <nwmigrat.h>

Function NWGetDefaultSupportModule
  (conn : NWCONN_HANDLE;
   supportModuleID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

supportModuleID

(OUT) Points to the currently supported module ID.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x89EC	NO_SUCH_SEGMENT

0x00F0	ERR_INVALID_SM_ID
0x89FB	N0_SUCH_PROPERTY

NCP Calls

0x2222 90 134 Get/Set Default Read-Write Support Module ID

See Also

NWSetDefaultSupportModule, **NWGetSupportModuleInfo**

NWGetDMFileInfo

Returns information about data migrated files
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Data Migration

Syntax

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

NWCCODE N_API NWGetDMFileInfo (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint8          nameSpace,
   pnuint32        supportModuleID,
   pnuint32        restoreTime,
   pnuint32        dataStreams);
```

Pascal Syntax

```
#include <nwmigrat.h>

Function NWGetDMFileInfo
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   nameSpace : nuint8;
   supportModuleID : pnuint32;
   restoreTime : pnuint32;
   dataStreams : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired name space (optional).

path

(IN) Points to a valid DOS path pointing to a file.

nameSpace

(IN) Specifies the name space of the DOS path.

supportModuleID

(OUT) Points to the ID of the Support Module containing the migrated data.

restoreTime

(OUT) Points to an estimate of the time (in ticks) needed to retrieve the data.

dataStreams

(OUT) Points to an array of supported data streams.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	Bad AFP Entry ID
0x899E	INVALID_FILENAME
0x89A8	ERR_ACCESS_DENIED
0x89BF	INVALID_NAME_SPACE
0x00F0	ERR_INVALID_SM_ID

NCP Calls

The time returned in the *restoreTime* parameter represents the estimated number of ticks needed. There are 18.2 ticks in one second.

NCP Calls

0x2222 87 06 Obtain File or Subdirectory Information

0x2222 90 129 DM File Information

See Also

NWGetSupportModuleInfo, **NWMoveFileFromDM**, **NWMoveFileToDM**

NWGetDMVolumeInfo

Returns information about the Data Migrator NLM on a NetWare volume
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Data Migration

Syntax

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

NWCCODE N_API NWGetDMVolumeInfo (
   NWCONN_HANDLE   conn,
   nuint16         volume,
   nuint32         supportModuleID,
   pnuint32        numberOfFilesMigrated,
   pnuint32        totalMigratedSize,
   pnuint32        spaceUsedOnDM,
   pnuint32        limboSpaceUsedOnDM,
   pnuint32        spaceMigrated,
   pnuint32        filesInLimbo);
```

Pascal Syntax

```
#include <nwmigrat.h>

Function NWGetDMVolumeInfo
  (conn : NWCONN_HANDLE;
   volume : nuint16;
   supportModuleID : nuint32;
   numberOfFilesMigrated : pnuint32;
   totalMigratedSize : pnuint32;
   spaceUsedOnDM : pnuint32;
   limboSpaceUsedOnDM : pnuint32;
   spaceMigrated : pnuint32;
   filesInLimbo : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volume

(IN) Specifies the volume number having the migrated files.

supportModuleID

(IN) Specifies the currently supported module ID.

numberOfFilesMigrated

(OUT) Points to the migrated number of files from the selected volume.

totalMigratedSize

(OUT) Points to the total number of bytes needed to recover all the data on the selected volume.

spaceUsedOnDM

(OUT) Points to the size of the data on the migrator media.

limboSpaceUsedOnDM

(OUT) Points to the size of the demigrated data on the migrator area. Since the data is generally Read Only, the file will be kept on the migrator until the file is either deleted or remigrated with changes.

spaceMigrated

(OUT) Points to the total size of the migrated data for the volume (includes the limbo space used).

filesInLimbo

(OUT) Points to the number of files that are in limbo or were demigrated with SAVE_KEY_WHEN_FILE_IS_DEMIGRATED and have not been migrated back to the data migrator.

Return Values

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

0x0000	SUCCESSFUL
0x00F0	ERR_INVALID_SM_ID
0x8801	INVALID_CONNECTION
0x8978	ERR_VOLUME_FLAG_NOT_SET
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x8998	VOLUME_DOES_NOT_EXIST

NCP Calls

0x2222 90 130 Get Volume DM Status

See Also

NWGetDefaultSupportModule, NWGetDataMigratorInfo, NWGetSupportModuleInfo

NWGetSupportModuleInfo

Returns information about the Data Migrator NLM support modules or a list of all loaded support module IDs

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Data Migration

Syntax

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

NWCCODE N_API NWGetSupportModuleInfo (
   NWCONN_HANDLE   conn,
   nuint32         informationLevel,
   nuint32         supportModuleID,
   pnuint8         returnInfo,
   pnuint32        returnInfoLen);
```

Pascal Syntax

```
#include <nwmigrat.h>

Function NWGetSupportModuleInfo
  (conn : NWCONN_HANDLE;
   informationLevel : nuint32;
   supportModuleID : nuint32;
   returnInfo : pnuint8;
   returnInfoLen : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

informationLevel

(IN) Specifies the level of information to be returned. If information Level = 0, returns information about the DM NLM support module; if information Level = 1, returns a list of all loaded support module IDs.

supportModuleID

(IN) Specifies the assigned ID number of the support module migrating the data.

returnInfo

(OUT) Points to the area in which to store the information.

returnInfoLen

(OUT) Points to the size of the data area the user allocated in which to return information.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x89A8	ERR_ACCESS_DENIED
0x00F0	ERR_INVALID_SM_ID
0x89FF	Failure, Invalid Info Level, or Invalid Parameter

Remarks

If the *informationLevel* parameter contains 0 (zero), the SUPPORT_MODULE_INFO structure will be used to return information about the DM NLM support module to the *returnInfo* parameter. If the *informationLevel* parameter contains 1, the SUPPORT_MODULE_IDS structure will be used to return a list of all loaded support module IDs to the *returnInfo* parameter.

NCP Calls

0x2222 90 132 DM Support Module Information

See Also

NWGetDefaultSupportModule, **NWGetDataMigratorInfo**, **NWGetDMVolumeInfo**

NWMoveFileFromDM

Moves file data from an on-line, long term storage medium to a NetWare volume

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Data Migration

Syntax

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

NWCCODE N_API NWMoveFileFromDM (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint8          nameSpace);
```

Pascal Syntax

```
#include <nwmigrat.h>

Function NWMoveFileFromDM
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   nameSpace : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired name space (optional).

path

(IN) Points to a valid DOS path pointing to a file.

nameSpace

(IN) Specifies the name space of the DOS path.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8978	ERR_VOLUME_FLAG_NOT_SET
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899E	INVALID_FILENAME
0x89A8	ERR_ACCESS_DENIED
0x89FB	Invalid Namespace (abends the server)

NCP Calls

0x2222 87 06 Obtain File or Subdirectory Information
0x2222 90 133 Move File Data From DM

See Also

NWMoveFileToDM, **NWSetDefaultSupportModule**, **NWGetDMFileInfo**

NWMoveFileToDM

Moves file data to an online, long term storage medium but leaves the file visible on a NetWare volume

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Data Migration

Syntax

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

NWCCODE N_API NWMoveFileToDM (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint8          nameSpace,
   nuint32         supportModuleID,
   nuint32         saveKeyFlag);
```

Pascal Syntax

```
#include <nwmigrat.h>

Function NWMoveFileToDM
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   nameSpace : nuint8;
   supportModuleID : nuint32;
   saveKeyFlag : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired name space (optional).

path

(IN) Points to a valid DOS path (pointing to a directory or file).

nameSpace

(IN) Specifies the name space of the DOS path.

supportModuleID

(IN) Specifies the assigned ID number of the support module migrating the data.

saveKeyFlag

(IN) Specifies if the migrator key will be saved when the file is demigrated: 0 = migrator key will not be saved, 1 = migrator key will be saved.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899E	INVALID_FILENAME
0x899C	INVALID_PATH
0x89A8	ERR_ACCESS_DENIED
0x89FB	Invalid Namespace

Remarks

If *saveKeyFlag* equals SAVE_KEY_WHEN_FILE_IS_DEMIGRATED, the key will be saved when the file is demigrated. This saves time because the file will not be deleted from the migrated media and will be checked for changes before subsequent migrations.

NCP Calls

0x2222 87 06 Obtain File or Subdirectory Information
0x2222 90 128 Move File Data To DM

See Also

NWMoveFileFromDM, **NWSetDefaultSupportModule**,

File Service Group

NWGetDMFileInfo

NWSetDefaultSupportModule

Sets the default Read/Write support module ID
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Data Migration

Syntax

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

NWCCODE N_API NWSetDefaultSupportModule (
   NWCONN_HANDLE   conn,
   pnuint32        supportModuleID);
```

Pascal Syntax

```
#include <nwmigrat.h>

Function NWSetDefaultSupportModule
  (conn : NWCONN_HANDLE;
   supportModuleID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

supportModuleID

(IN) Points to the support module ID.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x89EC	NO_SUCH_SEGMENT

0x89FB	N0_SUCH_PROPERTY or INVALID_PARAMETERS

NCP Calls

0x2222 90 134 Get/Set Default Read Write Support Module ID

File Service Group

Data Migration: Structures

SUPPORT_MODULE_IDS

Returns a list of support module IDs (level 1 information) by **NWGetSupportModuleInfo**

Service: Data Migration

Defined In: nwmigrat.h

Structure

```
typedef struct
{
   nuint32   numberOfSMs;
   nuint32   SMIDs[MAX_NUM_OF_SM];
} SUPPORT_MODULE_IDS;
```

Pascal Structure

```
Defined in nwmigrat.inc

  SUPPORT_MODULE_IDS = Record
    numberOfSMs : nuint32;
    SMIDs : Array[0..MAX_NUM_OF_SM-1] Of nuint32
  End;
```

Fields

numberOfSMs

Indicates the number of valid support module IDs returned by the Data Migrator.

SMIDs

Indicates the list of support module IDs.

SUPPORT_MODULE_INFO

Returns (level 0) support module information by **NWGetSupportModuleInfo**

Service: Data Migration

Defined In: nwmigrat.h

Structure

```
typedef struct
{
   nuint32   IOStatus;
   nuint32   InfoBlockSize;
   nuint32   AvailSpace;
   nuint32   UsedSpace;
   nuint8    SMInfo[MAX_SIZE_OF_SM_STRING + MAX_SIZE_OF_SM_INFO];
} SUPPORT_MODULE_INFO;
```

Defined In

```
Defined in nwmigrat.inc

  SUPPORT_MODULE_INFO = Record
    IOStatus : nuint32;
    InfoBlockSize : nuint32;
    AvailSpace : nuint32;
    UsedSpace : nuint32; (*A length preceded string is followed by SMIn
    SMInfo : Array[0..MAX_SIZE_OF_SM_STRING + MAX_SIZE_OF_SM_INFO - 1]
  End;
```

Fields

IOStatus

Indicates the IO read and write access status of the associated storage device .

InfoBlockSize

Indicates the information block size on the associated storage device.

AvailSpace

Indicates the amount of space available on the associated storage device.

UsedSpace

Indicates the amount of used space on the associated storage device. This length-preceded string is followed by *SMInfo* data.

SMInfo

Indicates the support-module specific data in the form of a

length-preceded string.

Deleted File

Deleted File: Guides

Deleted File: Concept Guide

File Purging and Recovery

File Purging and Recovery on NetWare 2.2 Servers

File Purging and Recovery on NetWare 3.11 and 4.x Servers

Salvaging Files: Example

Compatibility of Deleted File with File System

Summary of File Purging and Recovery Functions

Additional Links

Deleted File: Functions

Deleted File: Structures

Parent Topic:

File Overview

File Purging and Recovery

NetWare® servers retain deleted files in a recoverable state. The final deallocation of a deleted file is called purging. Deleted File Services include functions for purging and recovering deleted files. Two functions purge and recover deleted files on NetWare 2.2 and later servers:

NWPurgeDeletedFile

NWRecoverDeletedFile

Although the same functions can handle deleted files across different versions of NetWare, there are important differences between version 2.2 and versions 3.11 and 4.x.

File Purging and Recovery on NetWare 3.11 and 4.x Servers

File Purging and Recovery on NetWare 2.2 Servers

Parent Topic:

Deleted File: Guides

Deleted File: Examples

Salvaging Files: Example

This code example recovers a deleted file.

> **NOTE:** This example contains DOS specific screen functions that should be ignored on non-DOS platforms.

Salvaging Files

```
#define N_PLAT_DOS

#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <fcntl.h>
#include <share.h>
#include <direct.h>
#include <dos.h>
#include <string.h>
#include <conio.h>
#include <time.h>
#include <nwfile.h>
#include <nwdirect.h>
#include <nwaudit.h>
#include <nwfse.h>
#include <nwdel.h>
#include <nwmisc.h>
#include <ntypes.h>

main()
{
   NWCCODE          ccode;
   NWCONN_HANDLE    conn;
   int              commandChar, purgeAllFlag;
   nstr8            newFileName[100];
   char             fullPathName[200];
   NWDIR_HANDLE     dirHandle;
   pnuint32         iterHandle;
   pnuint32         volNum;
   pnuint32         dirBase;
   NWDELETED_INFO   entryInfo;
   nuint32          delTime;
   char             scanDirectory[100], *charP;
   long             nen;
```

```
nen = -1;
purgeAllFlag = 0;
printf("Directory to scan: ");
gets(scanDirectory);
if(!scanDirectory[0])
{
   scanDirectory[0] = '\\';
   scanDirectory[1] = 0;

}

clrscr();
printf("Filename      Size      Attr      Date&Time     Seq#\r\n");

gotoxy(0, 24);
printf("A - purge all; P - purge; S - salvage; <enter> - next file;
        X - exit");
gotoxy(0, 24);
while(TRUE)
{
   ccode = NWScanForDeletedFiles((NWCONN_HANDLE)&conn,
                  (NWDIR_HANDLE)&dirHandle, (pnuint32)&iterHandle,
                  (pnuint32)&volNum, (pnuint32)&dirBase,
                  &entryInfo);

   if(ccode == 0)
      break;

   delTime = entryInfo.deletedDateAndTime;
   strcpy(newFileName, (const char *)entryInfo.name);
   if((charP = strchr(newFileName, '.')) == NULL)
      charP = " ";
   *charP++ =0;  /* overwrite period with a zero */

   if(wherey() == 24)
   {
      gotoxy(0, 23);
   }
   printf(" %-8.8s %-3.3s %8d 0x%04x %s %4d\r\n",
          newFileName, charP, entryInfo.fileSize, entryInfo.attribut
          delTime, nen & 0xFFFFFFL);

   /* SetScreenRegionAttribute(wherey()-1, 1, 10); */
   /* underline current file */
   /* SetScreenRegionAttribute(wherey()-2, 1, 7);  */
   /* un-underline prev file */

   if(!purgeAllFlag)
   {
      commandChar - getch();
      if(commandChar == 'A')
```

```
                purgeAllFlag = 1;
            }

            if((commandChar == 'p') || (commandChar =='P') || purgeAllFlag)
            {
               strcpy(fullPathName, scanDirectory);
               if(scanDirectory[1])
                  strcat(fullPathName, "\\");
               strcat(fullPathName, (const char *)entryInfo.name);
               if((ccode = NWPurgeDeletedFile(conn, dirHandle,
                    (nuint32)iterHandle,(nuint32)volNum, (nuint32)dirBase,
                    (unsigned char *)fullPathName)) !=0)

                  printf("Could Not Purge File %s; error = %d\r\n",
                          fullPathName, ccode);

            }
            else if((commandChar == 's') || (commandChar == 'S'))
            {
               if(wherey() == 24)
               {
                  /* ScrollScreenRegionUp(2, 22); */
                  gotoxy(0, 23);
               }
               printf("New Filename: ");
               gets(newFileName);
               strcpy(fullPathName, scanDirectory);
               if(scanDirectory[1])
               strcat(fullPathName, "\\");
               strcat(fullPathName, (const char *)entryInfo.name);
               ccode = NWRestoreErasedFile(conn, dirHandle, fullPathName,
                        (char *)nen, newFileName);
               if(ccode !=0)
                  printf("Could Not Salvage File %s; error = %d\r\n",
                          entryInfo.name, ccode);
            }
            else if((commandChar == 'x') || (commandChar == 'X'))
            {
               break;
            }
         }
         return (0);
      }
```

Parent Topic:

Deleted File: Guides

Related Topics:

File Purging and Recovery

Deleted File: Concepts

Compatibility of Deleted File with File System

NetWare® contains important changes to the file system in versions after 2.15. These changes primarily affect trustee rights, file attributes, and purgeable files. For further information about these changes, see File System: Guides.

Although differences between overlapping functions are noted, developers need to be aware of compatibility issues affecting specific functions. To verify the compatibility of a function, see its reference in Deleted File: Functions.

Parent Topic:

Deleted File: Guides

File Purging and Recovery on NetWare 2.2 Servers

When a client erases a file on a NetWare® 2.2 server, the server marks the file for deletion but does not relocate it. You cannot scan for such files on NetWare 2.2 servers. To restore a file, you must know the file path and filename.

NetWare 2.2 servers hold only the most recently deleted file for each client and purge older files to reclaim disk and directory space. Performing a purge operation on a 2.2 server permanently removes all deleted files. Files marked for deletion are only recoverable until the client attempts to erase or create another file.

Parent Topic:

File Purging and Recovery

File Purging and Recovery on NetWare 3.11 and 4.x Servers

When a client erases a file on a NetWare® 3.11 or 4.x server, the server moves the file to a holding area in the directory structure of the volume. You can scan this area for deleted files by calling **NWScanForDeletedFiles** using a search pattern. Scanning deleted files returns file information for all

recoverable files in a specified directory. The file information can be used to restore deleted files using **NWRestoreErasedFile**. No prior knowledge of filenames is necessary.

When you purge files on a NetWare 3.11 or 4.x server, only the specified files are removed from the holding area. Other deleted files are not affected. Deleted files can remain on the server for an indefinite period. However, if the server must reclaim disk space, the files can be purged, after which they cannot be recovered.

Parent Topic:

File Purging and Recovery

Summary of File Purging and Recovery Functions

These functions handle the purging and recovery of deleted NetWare® files.

Function	Comment
NWPurgeDeletedFile	Removes recoverable files from a NetWare server.
NWPurgeErasedFiles	Purges all erased files on a NetWare server. This function is for a 2.2 NetWare server.
NWRecoverDeletedFile	Recovers deleted files from the NetWare server.
NWRestoreErasedFile	Recovers the specified erased file on NetWare servers. This is for a NetWare 2.2 server.
NWScanForDeletedFiles	Scans the specified directory for any deleted (salvageable) files.

Parent Topic:

Deleted File: Guides

Deleted File: Functions

NWPurgeDeletedFile

Removes recoverable files from a NetWare® server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Deleted File

Syntax

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

NWCCODE N_API NWPurgeDeletedFile (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   nuint32         iterHandle,
   nuint32         volNum,
   nuint32         dirBase,
   pnstr8          fileName);
```

Pascal Syntax

```
#include <nwdel.inc>

Function NWPurgeDeletedFile
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   iterHandle : nuint32;
   volNum : nuint32;
   dirBase : nuint32;
   fileName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle to purge.

dirHandle

(IN) Specifies the directory handle for the directory containing the file to purge (valid for 3.x and above only).

iterHandle

(IN) Specifies the sequence number returned by **NWScanForDeletedFiles** (valid for 3.x and above only).

volNum

(IN) Specifies the volume number returned by **NWScanForDeletedFiles** (valid for 3.11 and above only).

dirBase

(IN) Specifies the directory base number returned by **NWScanForDeletedFiles** (valid for 3.11 and above only).

fileName

(IN) Points to the name of the file to purge (valid for 3.0 and 3.1 only).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8985	NO_CREATE_DELETE_PRIVILEGES
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

For 2.x servers, all salvageable files on all volumes of the specified NetWare server are purged. For 3.x servers, only the specified file is purged.

For 3.x servers, **NWPurgeDeletedFile** is used in connection with **NWScanForDeletedFiles**. *iterHandle*, *volNum*, and *dirBase* are returned by **NWScanForDeletedFiles** and should not be modified prior to calling **NWPurgeDeletedFile**.

Although parameters may only be valid for some servers, each parameter must be filled. Valid parameters for **NWPurgeDeletedFile** on each platform are listed below:

2.x	**3.0 and 3.1**	**3.11**
conn	*conn*	*conn*
	dirHandle	*dirHandle*
	sequence	*iterHandle*
		volNum

		dirBase
	fileName	

NCP Calls

0x2222 22 16 Purge Deleted File
0x2222 23 17 Get File Server Information
0x2222 87 18 Purge Salvageable File
0x2222 22 29 Purge Salvageable File

See Also

NWScanForDeletedFiles

NWPurgeErasedFiles

Purges all erased files on the specified NetWare server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Deleted File

Syntax

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

NWCCODE N_API NWPurgeErasedFiles (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwdel.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle to purge.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8981	NO_MORE_FILE_HANDLES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	Disk Map Error
0x89A1	DIRECTORY_IO_ERROR
0x89FB	ERR_NCP_NOT_SUPPORTED

0x89FF	Failure

Remarks

NWPurgeErasedFiles permanently deletes all files that have been marked for deletion and deallocates their space.

NCP Calls

0x2222 22 16 Purge Erased Files (old)

NWRecoverDeletedFile

Recovers deleted files from the NetWare server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Deleted File

Syntax

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

NWCCODE N_API NWRecoverDeletedFile (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   nuint32         iterHandle,
   nuint32         volNum,
   nuint32         dirBase,
   pnstr8          delFileName,
   pnstr8          rcvrFileName);
```

Pascal Syntax

```
#include <nwdel.inc>

Function NWRecoverDeletedFile
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   iterHandle : nuint32;
   volNum : nuint32;
   dirBase : nuint32;
   delFileName : pnstr8;
   rcvrFileName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle containing the deleted file.

dirHandle

(IN) Specifies the directory handle of the directory containing the file to recover.

iterHandle

(IN) Specifies the number returned by **NWScanForDeletedFiles** (valid for 3.x and above only, use NULL for 2.x).

volNum

(IN) Specifies the number returned by **NWScanForDeletedFiles** (valid for 3.11 and above only, use NULL for 2.x).

dirBase

(IN) Specifies the number returned by **NWScanForDeletedFiles** (valid for 3.11 and above only, use NULL for 2.x).

delFileName

(IN/OUT) Points to the name of the erased file:

```
IN    3.0, 3.1
OUT   2.x
```

rcvrFileName

(IN/OUT) Points to the name to use in recovering the file.

```
IN    3.0, 3.1
OUT   2.x
```

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x8984	NO_CREATE_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FE	File name already exists in this directory
0x89FF	Failure

Remarks

For 3.x-4.x servers, files deleted by a client are moved to a holding area on the volume until they are either purged, restored (by calling

NWRecoverDeletedFile), or replaced by other deleted files.

For 3.11 servers, the recovery is performed one file at a time. **NWRecoverDeletedFile** can also recover the deleted file and give it a new name. This feature alleviates problems with recovering a file when a new file exists with the same name.

For 2.x servers, applications are unable to scan for the deleted file. Therefore, *dirHandle* must contain a handle pointing to the directory containing the deleted file. If a file with the same name already exists in that same directory, the server gives the recovered file a new name and returns it to the caller.

For 2.x servers, only the last deleted entry is recoverable. Also, this file is recoverable only until a client attempts another erase or file create request. **NWScanForDeletedFiles** returns all necessary information to be passed into **NWRecoverDeletedFile**.

For 2.x servers, the server returns*delFileName* (rather than passing it in). This buffer cannot be NULL.

A 2.x NetWare server returns the appropriate name for *rcvrFileName*. For 3.x, the application must specify the file name in *rcvrFileName* not the path; no wildcards are allowed.

NOTE: Due to earlier support for 14 character names in NetWare, both *delFileName* and *rcvrFileName* buffers must be at least 15 bytes long.

Although parameters may only be valid for some servers, each parameter must be filled. Valid parameters for **NWRecoverDeletedFile** on each platform are listed below:

2.x	**3.0 and 3.1**	**3.11 and 4.0**
conn	*conn*	*conn*
dirHandle	*dirHandle*	*dirHandle*
	sequence	*iterHandle*
		volNum
		dirBase
deletedFileName(return)	*deletedFileName*(passed in)	
recoverFileName(return)	*recoverFileName* (passed in)	*rcvrFileName*

NCP Calls

0x2222 22 17 Recover Erased File (old)
0x2222 22 28 Recover Salvageable File

0x2222 23 17 Get File Server Information
0x2222 87 17 Recover Salvageable File

See Also

NWScanForDeletedFiles

NWRestoreErasedFile

Recovers the specified erased file on NetWare servers
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Deleted File

Syntax

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

NWCCODE N_API NWRestoreErasedFile (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          dirPath,
   pnstr8          oldName,
   pnstr8          newName);
```

Pascal Syntax

```
#include <nwdel.inc>

Function NWRestoreErasedFile
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   dirPath : pnstr8;
   oldName : pnstr8;
   newName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle containing the erased file.

dirHandle

(IN) Specifies the directory handle of the directory containing the file to restore.

dirPath

(IN) Points to the path (relative to *dirHandle*) containing the erased file.

oldName

(IN) Points to the original name of the erased file.

newName

(IN) Points to the name to be given to the restored file.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89FB	ERR_NCP_NOT_SUPPORTED

Remarks

dirHandle can be zero if *dirPath* contains the complete path, including the volume name.

Applications can optionally use *newName* to rename the file when restoring.

NCP Calls

0x2222 22 17 Recover Erased File (old)

NWScanForDeletedFiles

Scans the specified directory for any deleted (salvageable) files
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Deleted File

Syntax

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

NWCCODE N_API NWScanForDeletedFiles (
   NWCONN_HANDLE           conn,
   NWDIR_HANDLE            dirHandle,
   pnuint32                iterHandle,
   pnuint32                volNum,
   pnuint32                dirBase,
   NWDELETED_INFO N_FAR  *entryInfo);
```

Pascal Syntax

```
#include <nwdel.inc>

Function NWScanForDeletedFiles
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   iterHandle : pnuint32;
   volNum : pnuint32;
   dirBase : pnuint32;
   Var entryInfo : NWDELETED_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle of the directory to scan.

iterHandle

(IN) Points to the address of the search sequence number. Must be initially set to -1.

volNum

(OUT) Points to the volume's number index (valid for 3.11 and above only).

dirBase

(OUT) Points to the directory's number index (valid for 3.11 and above only).

entryInfo

(OUT) Points to NWDELETED_INFO, containing the deleted file information.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x899B	BAD_DIRECTORY_HANDLE
0x89FF	No more salvageable files in directory

Remarks

NWScanForDeletedFiles replaces **NWScanSalvageableFiles**.

Initially, *iterHandle* needs to be set to -1. The server maintains the sequence number once a match has been found. No file names or wildcards are allowed in the search.

volNum and *dirBase* are used only when scanning NetWare 3.11 and above. These two numbers are indices used by the server to speed up the location of a deleted file. They should not be modified by an application.

Although parameters may only be valid for some servers, each parameter must be filled. The valid parameters for **NWScanForDeletedFiles** on each platform follow:

3.0 and 3.1	**3.11**
conn	*conn*
dirHandle	*dirHandle*
sequence	*iterHandle*
	volNum
	dirBase

NCP Calls

0x2222 22 27 Scan Salvageable Files
0x2222 23 17 Get File Server Information
0x2222 87 16 Scan Salvageable Files

See Also

NWPurgeDeletedFile, **NWRecoverDeletedFile**

Deleted File: Structures

NWDELETED_INFO

Returns information on a deleted file
Service: Deleted File
Defined In: nwdel.h

Structure

```
typedef struct
{
   nuint32   sequence;
   nuint32   parent;
   nuint32   attributes;
   nuint8    uniqueID;
   nuint8    flags;
   nuint8    nameSpace;
   nuint8    nameLength;
   nuint8    name[256];
   nuint32   creationDateAndTime;
   nuint32   ownerID;
   nuint32   lastArchiveDateAndTime;
   nuint32   lastArchiverID;
   nuint32   updateDateAndTime;
   nuint32   updatorID;
   nuint32   fileSize;
   nuint8    reserved[44];
   nuint16   inheritedRightsMask;
   nuint16   lastAccessDate;
   nuint32   deletedTime;
   nuint32   deletedDateAndTime;
   nuint32   deletorID;
   nuint8    reserved3[16];
} NWDELETED_INFO;
```

Pascal Structure

```
Defined in nwdel.inc

  NWDELETED_INFO = Record
    sequence : nuint32;
    parent : nuint32;
    attributes : nuint32;
    uniqueID : nuint8;
    flags : nuint8;
    nameSpace : nuint8;
    nameLength : nuint8;
    name : Array[0..255] Of nuint8;
    creationDateAndTime : nuint32;
    ownerID : nuint32;
```

```
    lastArchiveDateAndTime : nuint32;
    lastArchiverID : nuint32;
    updateDateAndTime : nuint32;
    updatorID : nuint32;
    fileSize : nuint32;
    reserved : Array[0..43] Of nuint8;
    inheritedRightsMask : nuint16;
    lastAccessDate : nuint16;
    deletedTime : nuint32;
    deletedDateAndTime : nuint32;
    deletorID : nuint32;
    reserved3 : Array[0..15] Of nuint8
  End;
```

Fields

sequence

Indicates the sequence number of the associated information.

parent

Indicates the ID of the owning subdirectory.

attributes

Indicates the attributes of the associated file.

uniqueID

Indicates the entry number of the file.

flags

Indicates the DOS attributes on the deleted file.

nameSpace

Indicates the name space of the associated file:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_OS2
4 NW_NS_LONG

nameLength

Indicates the length of the file name.

name

Indicates the file name.

creationDateAndTime

Indicates the date and time the file was created.

ownerID

Indicates the object which created the file.

lastArchiveDateAndTime

Indicates the date and time the file was last archived.

lastArchiverID

Indicates the object which last archived the file.

updateDateAndTime

Indicates the date and time the file was last updated.

updatorID

Indicates the object which last updated the file.

fileSize

Indicates the size of the file in bytes.

reserved

Is reserved for future use.

inheritedRightsMask

Indicates a bit mask of the following:

0x0000 TR_NONE
0x0001 TR_READ
0x0002 TR_WRITE
0x0004 TR_OPEN
0x0004 TR_DIRECTORY
0x0008 TR_CREATE
0x0010 TR_DELETE
0x0010 TR_ERASE
0x0020 TR_OWNERSHIP
0x0020 TR_ACCESS_CTRL
0x0040 TR_FILE_SCAN
0x0040 TR_SEARCH
0x0040 TR_FILE_ACCESS
0x0080 TR_MODIFY
0x01FB TR_ALL
0x0100 TR_SUPERVISOR
0x00FB TR_NORMAL

lastAccessDate

Indicates the date the file was last accessed.

deletedTime

Indicates the time the file was deleted.

deletedDateAndTime

Indicates the date and time the file was deleted.

deletorID

Indicates the user ID of the person who deleted the file.

reserved3

Is reserved for future use.

Direct File System

Direct File System: Guides

Direct File System: Task Guide

Direct File System: Concept Guide

Direct File System: Structures

Parent Topic:

Direct File System: Guides

Direct File System: Tasks

Creating a File with the Direct File System

Files can be created by calling **DFScreat**. This creates a file on the specified volume with the permissions indicated, and leaves the file open in Direct File Mode (the file must later be closed). **DFScreat** is the DFS equivalent of **creat**.

Create files by calling **DFSsopen** (providing that the file does not currently exist). **DFSsopen** is the DFS equivalent of **sopen**.

Parent Topic:

Direct File System File Allocation

Extending a File Using Default File Allocation

Extend an NLM application file using default file allocation:

1. **Determine if space is available.**

   Call **DFSReturnVolumeMappingInformation** to obtain the volume size, allocation unit size, sector size, number of free blocks, number of blocks available in deleted files, number of blocks not available in deleted files, and so forth. If the desired number of blocks is not available on the volume, skip to Step 4.

   Since the OS is multitasking, it is possible (and likely) that other processes allocate (and free) volume blocks at any time, making information returned by **DFSReturnVolumeMappingInformation** obsolete. As a result you must design DFS NLM applications to repeat this procedure multiple times to deal with the failure of **DFSExpandFile** and **DFSFreeLimboVolumeSpace**.

2. **Determine the current allocation of the file, specifically the file block where the file is to be extended.**

   Call **DFSReturnFileMappingInformation** to determine the file's current mapping, including volume segments and blocks allocated. The starting file block to be extended must be specified in the extend request made in Step 3.

   A file might not be extended by specifying file block addresses that already exist. This means that an extend request to allocate file space for

a hole in a sparse file must not specify a number or contiguous blocks that exceed the size of the hole.

3. **Allocate blocks to expand the file.**

   Call **DFSExpandFile** specifying the number of blocks to be extended and wildcards (-1) for the volume block number and optionally for the volume segment number. This allows DFS to select the range of contiguous blocks used to extend the file.

   For normal files, extending a file a single block at a time and specifying wildcards for **both** the volume block address and the volume segment is identical in function to letting the NetWare® OS allocate for normal files.

   Specifying a wildcard segment number in conjunction with a wildcard volume block address allows the OS to alternate its selection of volume segments for file allocation, thus facilitating file striping on systems where it is advantageous to stripe files. Specifying a larger number of blocks with a wildcard volume segment stripes the file (provided that multiple volume segments exist) with the larger granularity. If there is not enough contiguous available space to expand the file by the requested number of blocks, applications may be required to make several calls specifying smaller request sizes, or fail the request.

   If the return code indicates that the above operation was not successful, proceed to Step 4. Otherwise, the file has been extended as requested.

4. **Free up volume space.**

   Call **DFSFreeLimboVolumeSpace** specifying the volume number and the requested number of blocks to be freed. This causes one or more deleted files to be purged from the volume in order of time of deletion. Go to Step 1.

Parent Topic:

Extending an NLM File

Extending a File Using Specific File Allocation

Extend an NLM application file by selecting the volume segments and/or blocks to be allocated:

1. **Determine if space is available.**

   Call **DFSReturnVolumeMappingInformation** to obtain the volume size, allocation unit size, sector size, number of free blocks, number of blocks available in deleted files, number of blocks not available in deleted files, and so forth. If adequate space for the file extension is not available, go to Step 5.

2. **Determine the current allocation for the file.**

   Call **DFSReturnFileMappingInformation** to determine the file's current mapping, including volume segments and blocks allocated. This information is required to determine where to extend a file.

   A file may not be extended by specifying file block addresses which already exist. This means that an extend request to allocate file space for a hole in a sparse file must not specify a number or contiguous blocks which exceed the size of the hole.

3. **Determine the available blocks on a volume.**

   Call **DFSReturnVolumeBlockInformation** to obtain a bit map of available blocks on the desired volume. Determine from the bitmap a contiguous range of blocks large enough to extend the file as needed.

   As a result of multitasking, the bitmap of available blocks is valid only at the moment it is obtained, and may have changed by the time an application NLM requests a specific range of blocks to be allocated for a file (possibly requiring this step to be repeated multiple times).

   Call **DFSExpandFile** to specify the range of contiguous available blocks selected from the preceding bitmap. Specifying more than a single block causes the requested contiguous blocks to be allocated from the same volume segment. A good return code indicates that the requested function is completed.

4. **Extend the file.**

   If enough contiguous space to expand the file the requested number of blocks is not available, applications may be required to make several calls specifying smaller request sizes, or fail the request.

   If the return code indicates that the above operation was not successful, proceed to Step 5. Some other module may have allocated the requested blocks, so the calling application must be prepared to retry the operation several times.

5. **Free up volume space.**

   Call **DFSFreeLimboVolumeSpace** to specify the volume number and the requested number of blocks to be freed. This causes one or more deleted files to be purged from the volume in the order of the time of deletion. Go back to Step 1.

Parent Topic:

Extending an NLM File

Extending an NLM File

Follow one of these options to extend an NLM application file:

Follow one of these options to extend an NLM application file:

Extending a File Using Default File Allocation

Extending a File Using Specific File Allocation

Parent Topic:

Direct File System: Guides

Direct File System: Concepts

Direct File System Completion Codes

The following table lists and defines DFS completion codes.

Table auto. DFS Completion Codes

Code	Name	Meaning
0	DFSNormalCompletion	The operation was completed as specified.
1	DFSInsufficientSpace	The required space does not currently exist on the volume to expand the file as requested.
4	DFSVolumeSegmentDeactivated	The volume segment(s) on which the file is located have been deactivated by the operating system.
16	DFSTruncationFailure	The **DFSSetEndOfFile** function detected that the caller requested excess blocks to be truncated, but the file was open for other connections, causing the request to be rejected.
17	DFSHoleInFileError	An operation (read or write) was requested for one or more file sectors where file space has not been allocated (a sparse file). The buffer is also zeroed for read functions.
18	DFSParameterError	The function caller supplied an invalid parameter.
19	DFSOverlapError	An attempt was made to allocate additional file space where file blocks already exist.
20	DFSSegmentError	A volume segment number was requested that was not one of the volume segments of the volume.
21	DFSBoundaryError	One or more blocks in the range requested are not currently

		available, or are not in the volume or volume segment specified.
22	DFSInsufficientLimboFileSpace	The request could not be completed because there were not enough contiguous limbo blocks to complete the request successfully.
23	DFSNotInDirectFileMode	A function requiring a file to be opened in direct mode (using **DFSsopen** or **DFScreat**) was requested, but the file is not currently opened in direct mode.
24	DFSOperationBeyondEndOfFile	A read operation was requested beyond the end-of-file (current file size).
129	DFSOutOfHandles	All available handles for the file are already in use.
131	DFSHardIOError	Problem decompressing or insufficient allocatable space.
136	DFSInvalidFileHandle	A DFS call was made using a file handle that is not valid---typically an open was omitted or the user inadvertently closed the file before attempting this access.
147	DFSNoReadPrivilege	The current connection does not have read privileges for the file.
148	DFSNoWritePrivilege	The current connection does not have write privileges for the file.
149	DFSFileDetached	The file system will not allow further processing on this handle.
150	DFSInsufficientMemory	The Direct File System could not obtain sufficient memory to complete the requested function.
152	DFSInvalidVolume	The volume number specified does not exist or not mounted.
-1	DFSFailedCompletion	The requested operation was not completed.

Parent Topic:

Direct File System: Guides

Direct File System File Allocation

The DFS allocation functions allow a great deal of control in determining where and how file space is allocated. The application NLM may allow the OS to allocate required space for a file using OS default allocation, or may request allocation to be on a specific volume segment and/or specify the actual volume blocks to be allocated. File allocation may be used to fill holes in sparse files or to extend existing files. Writing to a nonexistent area in a file (either to a hole or beyond the allocated file space) is not allowed with the DFS functions.

Creating a File with the Direct File System

Extending an NLM File

Impact of Striping

Setting the File Size and Zero-Filling with DFS

Parent Topic:

Direct File System: Guides

Direct File System File Locks

All DFS application file, record, and field locks must be provided and managed by the DFS application NLM. OS utilities that are designed to perform reorganization or relocation of DFS files use an exclusive lock on the file, so that the lock fails if the file is currently in use by an application NLM. An application NLM need only lock the file with a shared or nonexclusive lock to ensure that an OS utility NLM has not exclusively locked the file for operations such as reorganization.

An application NLM should make sure that the current connection ID matches the connection ID of the client so that OS Auditing is meaningful. However, auditing of database record and field accesses require that the database application NLM perform its own auditing, since DFS is not aware of the actual record and field definitions.

Parent Topic:

Direct File System: Guides

Direct File System Functions

Table auto. Direct File System Functions

Function	Purpose
DFSclose	Closes a file that is open in direct file mode.

DFScreat	Creates and opens a file in direct file mode, and returns a file handle.
DFSExpandFile	Expands a file with a range of contiguous blocks.
DFSFreeLimboVolumeSpace	Frees a number of limbo blocks on a volume.
DFSRead	Reads sectors from a file in direct file mode (sleeps until completion).
DFSReadNoWait	Reads sectors from a file in direct file mode (returns immediately after initiation).
DFSReturnFileMappingInformation	Returns file extents, each with number of blocks and starting file and volume block numbers.
DFSReturnVolumeBlockInformation	Returns volume block usage bitmap.
DFSReturnVolumeMappingInformation	Returns information about a volume required for file allocation.
DFSSetEndOfFile	Sets the file size of a file.
DFSsopen	Opens a file in direct file mode.
DFSWrite	Writes sectors into a file using DFS (sleeps until completion).
DFSWriteNoWait	Writes sectors into a file using DFS (returns immediately after initiation).

Parent Topic:

Direct File System: Guides

Direct File System I/O

An open file is in one of two possible modes: normal mode or direct mode. In a normal open mode, any of the valid opens are used. For direct open mode, the **DFSsopen** function is used. If a file is already open in normal mode and **DFSsopen** is called, the file changes from the normal mode to the direct mode.

In this condition, any files in the normal open mode can continue to be read, but attempts to write to the file fail. Programs using the direct mode can

read and write to the file successfully. The only way to write to files in the normal open mode again is by closing all direct mode opens, and closing and re-opening the normal mode open.

When a file is successfully opened for direct file I/O by calling **DFSsopen**, the server opens the file, flushes all cache entries for the file, and flags the file so that future I/Os do not use caching and TTS functions. Subsequent file I/O must be done using the I/O functions **DFSRead**, **DFSReadNoWait**, **DFSWrite**, or **DFSWriteNoWait**. The "no wait" versions of these functions do not block execution of the thread until completion, thus allowing a single thread to have multiple outstanding DFS I/O functions.

When direct file I/O operations are completed, the file must be closed by calling **DFSclose**. The file may not be used for normal I/O writes until all handles are relinquished for the file by calling **DFSclose**, followed by a normal open. If a normal open exists, it must be closed and re-opened for read/writes.

When a file is in Direct mode, writes may not be made to areas of the file where a hole exists (sparse files) or beyond the file's currently allocated last block address. Read operations to these areas are indicated successful and the data area is zeroed.

Also, a file cannot be extended while in Direct mode by writing beyond its current extents. A separate call to **DFSExpandFile** must be made to extend a file or to fill in holes in a file area.

Parent Topic:

Direct File System: Guides

Related Topics:

Direct File System File Locks

Direct File System Overview

NetWare® Direct File System (DFS) functions provide a method of bypassing the NetWare disk caching and Transaction Tracking System™ (TTS™) subsystems. The NetWare Cache and NetWare TTS provide the best possible throughput and **should not** normally be bypassed. However, several applications exist where this bypass may be desirable:

Large database packages typically provide their own caching and transaction tracking facilities, tailored to specific requirements and integrated into the package, making it desirable to use their caching and transaction facilities instead of those provided by NetWare. (Performance degradation results when both are used together.)

Backup applications often access large amounts of data not being accessed by other applications. If these accesses are made through the cache, the cache becomes non-relevant for all other accesses, and general

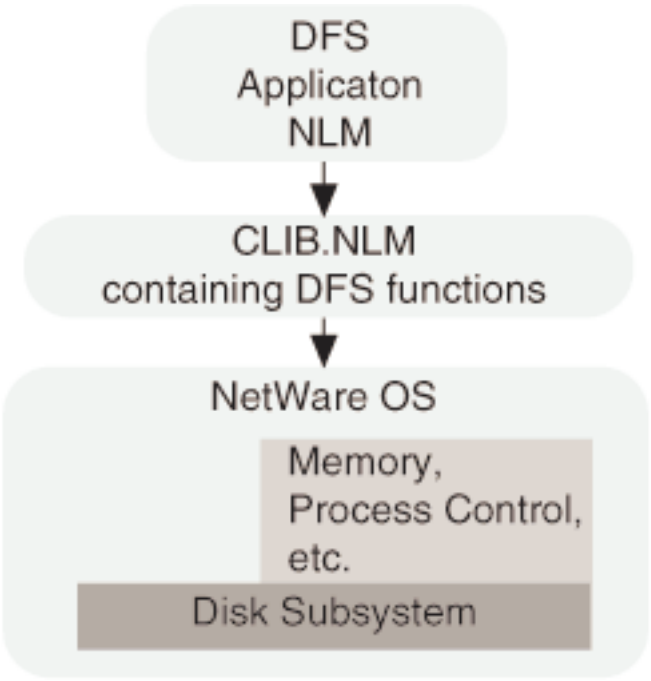
server performance suffers for other users.

Some utilities may also require the ability to specify exactly where files are placed so that volumes can be defragmented and files accessed optimally.

The DFS functions provide a standardized solution by bypassing the NetWare disk caching and TTS subsystems, while providing more direct control of file allocation. The DFS functions fall into two groups: basic direct mode file I/O, and file allocation primitives.

The following figure illustrates the DFS as it interfaces with the file system, applications, NLM™ applications, and with other portions of the NetWare OS.

Figure 1. Direct File System Interfaces



Parent Topic:

Direct File System: Guides

File Structures

Files in direct file mode may be viewed as an array of sectors numbered from zero to **n**, where **n** is the last sector in the last block currently allocated for the file. All direct file I/O must be done in multiples of sectors (one or

more). The sectors allocated in the file are actually allocated on an allocation block size, which is either 4 K, 8 K, 16 K, 32 K, or 64 K with NetWare® (the default allocation block size for NetWare 3.x is 4 K). The allocation size must be specified for a volume when the volume is created.

It is possible to create files that have blocks allocated for high addresses, but do not have blocks allocated for intermediate addresses. (Such files are called **sparse files**, and have holes in their actual allocation space). Normal (non-direct) I/O allows writes to be made into these holes or beyond the current end of allocated space for a file, in which case space is automatically allocated to fill the hole or extend the file. However, the Direct File System does not allow such writes when the file is in direct mode. Files must be extended before writes can be issued to holes or beyond the end of the allocated space for a file.

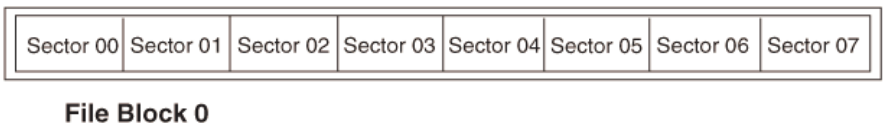
The logical block address of the blocks in the above array are referred to as **File Block Addresses**, and the associated logical sector addresses are referred to as **File Sector Addresses**. These addresses exist even for holes in the file, though actual storage space may not have been allocated for the corresponding locations on the file.

No space is allocated when a file is initially created by **DFScreat** or **DFSsopen**. All space must be allocated by the application process for files in direct mode (and must be allocated on the same volume).

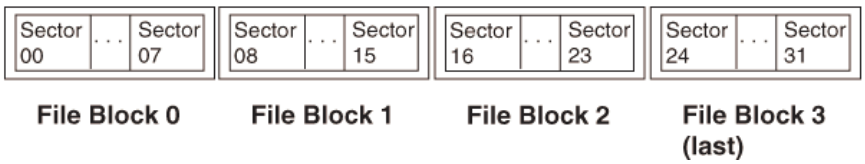
When a file not currently opened by another process is opened in direct mode, the OS creates a turbo FAT for the file if one does not currently exist, flushes the cache of entries for the file, and marks the file as open in direct mode (only direct mode operations are allowed).

Sample File Structures

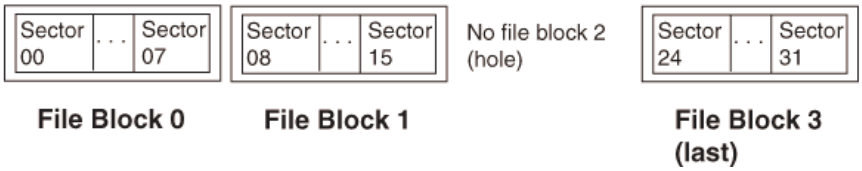
File with 4 K allocation block size and only one block allocated:



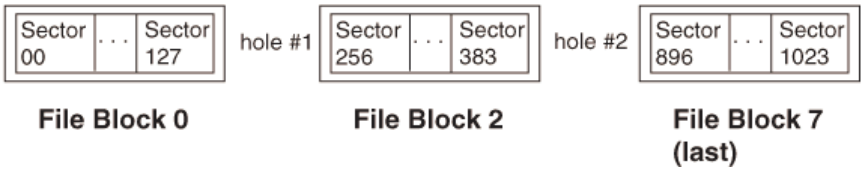
File with 4 K allocation block size and four blocks allocated (no holes):



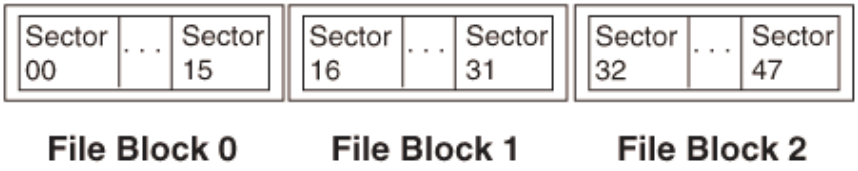
File with 4 K allocation block size and three blocks allocated (with hole):



File with 64 K allocation and three blocks allocated (with hole):



File with 8 K allocation block size and three blocks allocated (no holes):



Parent Topic:

Direct File System: Guides

Impact of Striping

Causing a file to be allocated with striping has different effects on different configurations. A drive array normally provides optimum throughput using internal striping, so specifying striping by the OS in this case would defeat optimum throughput, while specifying striping on files on a SCSI host adapter providing disconnect with multiple drives normally provides more optimal performance. Striping may not necessarily provide significant performance benefits for extremely large files accessed in true random fashion, but provides performance benefits when accessed sequentially.

Parent Topic:

Direct File System File Allocation

Setting the File Size and Zero-Filling with DFS

The size of a file may be specified independent of the actual file space allocated by making a call to **DFSSetEndOfFile**. Reads attempted beyond the current end-of-file (indicated by the file size) are always rejected as an attempted operation beyond the current file size.

If the file size is expanded beyond its previous value, the additional file area incorporated in the new file size is zero-filled, provided that the file space is actually allocated. If the file size specified is smaller than the previously indicated file size and the *returnTrunctatedBlocksFlag* is nonzero, blocks previously allocated beyond the newly defined file size are truncated, that is, returned to the OS for future use.

The file size is also modified by a call to **DFSWrite** (only the "wait" version) indicating a write to a file sector address beyond the current file size (the file size is updated accordingly) but within the range of blocks allocated for the file. If the write does not start immediately following the current file size, the intervening blocks are zero-filled.

When a file has additional space allocated at the end of the file by a call to **DFSExpandFile**, the additional file space is not zero-filled immediately. Subsequent writes to the file set a new file size and eliminate the requirement to zero-fill the additional file space. Calling **DFSSetEndOfFile** also sets a new file size, and zero-fills any sectors in the new file size that are beyond the previous file size.

When a file has additional space allocated to fill a hole in a sparse file, the **DFSExpandFile** function also zero-fills the additional space if the space is not beyond the current file size.

Parent Topic:

Direct File System File Allocation

Volume Structures

NetWare® volumes may be viewed as an array of allocation blocks numbered from zero to **n**, where **n** is the total number of allocation blocks in all segments of the volume (volume segments), minus one. Volumes may be extended by adding additional segments, which are logically added at the end of the list of current volume segments. Each logical block number in the logical array of volume blocks is referred to by a **Volume Block Number**. There are no holes in the volume block numbers. (Using this organization, a specific Volume Block Number also indirectly indicates the segment upon

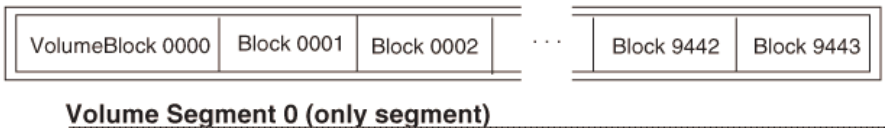
which it occurs in the list of volume segments.)

NetWare supports a maximum of 255 volumes, with a NetWare volume consisting of from 1 to 64 segments.

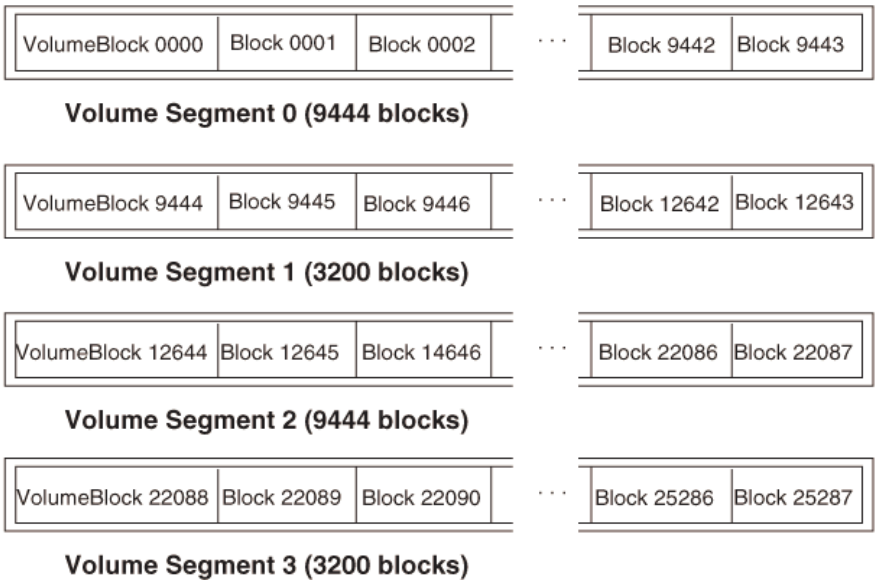
Multiple Volume Segments may exist on a single logical NetWare partition on a drive. It is also possible that a drive may have a single volume segment on it. A Logical partition has blocks numbered logically from zero through the last block available in the partition, and does not include the physical partition blocks which are allocated for Hot Fix™ and Mirroring tables at the time the NetWare partition is created.

Sample Volume

Volume with a single volume segment (9444 volume blocks):



Volume with four volume segments (25288 volume blocks):



Parent Topic:

Direct File System: Guides

Direct File System: Functions

DFSclose

Closes a file currently open in Direct File Mode
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSclose (
   LONG   fileHandle);
```

Parameters

fileHandle

(IN) The file handle returned from a prior successful call to **DFSsOpen** (the file must have previously opened by **DFSsOpen**). After a file is closed, the file handle is no longer valid and should not be reused.

Return Values

0	(0x00)	DFSNormalCompletion	The operation was completed as specified.
-1		DFSFailedCompletion	An error occurred closing the file. If this status is returned, *errno* is set to: 4 EBADF (Bad file number). If the function does not complete successfully, *NetWareErrno* is set.

Remarks

Calling **DFSclose** causes the file to be closed (the handle becomes invalid). If **DFSClose** determines that this was the last valid handle (no other opens outstanding for the file), the Direct File Mode flag is reset, allowing a subsequent open file call to be either **open** (normal mode), or **DFSsOpen** (direct mode). Remember, if a file is in direct mode, any programs with normal mode opens into the file are able to read the file but not write to it (see Direct File System I/O).

File Service Group

See Also

DFSsopen

DFScreat

Creates and opens a file in Direct File Mode, returning a file handle to the called file

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x

SMP Aware: No

Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFScreat (
   BYTE   *fileName,
   LONG   access,
   LONG   flagBits);
```

Parameters

fileName

(IN) Specifies the name of the file to be created. The filename must be NULL-terminated and must include the path, including the volume name but not the server name.

access

(IN) Specifies the access permissions for the file.

flagBits

(IN) Specifies the following when a file is created:

0x0001	DELETE_FILE_ON_CREATE_BIT
0x0002	NO_RIGHTS_CHECK_ON_CREATE_BIT

Return Values

!=	DFSFailedCompletion	The file now exists, is open, and is in direct mode. The return value is the file handle assigned when the file was created.

!= -1	RemarksDFSFailedCompletion	An error occurred creating the file.

If -1 is returned, *errno* is set to

1	ENONENT	No such file.
6	EACCES	Permission denied.
9	EINVAL	Invalid argument.

If the function does not complete successfully, *NetWareErrno* is set to

152	(0x98)	ERR_INVALID_VOLUME
156	(0x9C)	ERR_INVALID_PATH

Remarks

Calling **DFScreat** causes DFS to create a file, or to truncate the file if it already exists **and** if the current connection has write privileges. The name of the file to be created is given by the *filename* parameter. If the file exists, it is truncated to contain no data and the preceding permission setting is unchanged. The file is switched to direct mode, forcing subsequent file accesses to be direct (The file must be extended using **DFSExpandFile** to provide required file space). The file is left open and must be closed by a subsequent call to **DFSclose**.

Not all functions are allowed with this form of open once the file has been created. If additional functions such as specifying a stream are required, the caller should close the file and open it again by calling **DFSsopen**.

The *access* permissions are defined in FCNTL.H as follows:

0x0000	O_RDONLY	open for read only now if this wrap
0x0001	O_WRONLY	open for write only
0x0002	O_RDWR	open for read and write
0x0010	O_APPEND	writes done at end of file
0x0020	O_CREAT	create new file if one does not exist

0x0040	O_TRUNC	truncate existing file
0x0080	O_EXCL	exclusive open

If *access* is 0, the default value is O_CREAT, O_TRUNC, and O_WRONLY.

See Also

DFSclose, **DFSExpandFile**, **DFSsopen**

DFSExpandFile

Requests DFS to expand a file with a range of contiguous blocks
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSExpandFile (
   LONG   fileHandle,
   LONG   fileBlockNumber,
   LONG   numberOfBlocks,
   LONG   volumeBlockNumber,
   LONG   segmentNumber);
```

Parameters

fileHandle

(IN) The file handle returned from a prior **DFSsopen** or **DFScreat** call.

fileBlockNumber

(IN) The beginning file logical block number where the additional contiguous space is to be allocated.

numberOfBlocks

(IN) The number of contiguous blocks requested to be linked into the file allocation at the starting location.

volumeBlockNumber

(IN) The beginning volume logical block number at which contiguous blocks are to be allocated for the file. A wildcard value of -1 indicates that DFS can allocate the blocks anywhere it can find the required contiguous space on the volume segment.

segmentNumber

(IN) The volume segment number where the contiguous blocks are to be allocated when the logical volume block number is not specified (a wildcard volume block number was provided). A wildcard value of -1 in this parameter indicates that DFS can allocate the blocks in any volume segment on the volume where the specified number of contiguous free blocks can be found.

Return Values

0	DFSNormalCompletion	File was expanded in the area specified.
1	DFSInsufficientSpace	The required space does not exist on the volume to expand the file as requested.
18	DFSParameterError	The caller supplied one or more invalid parameters.
19	DFSOverlapError	An attempt was made to allocate additional file space where file blocks already exist.
20	DFSSegmentError	A volume segment was specified which does not exist on the volume.
21	DFSBoundaryError	One or more blocks in the range requested are not available or are not in the volume or volume segment specified.
131	DFSHardIOError	Attempted allocation of file blocks where file blocks are already defined, etc.
136	DFSInvalidFileHandle	A DFS call was made using a file handle which is not valid---typically an open was omitted or the user inadvertently closed the file before attempting this access.
148	DFSNoWritePrivilege	The current connection does not have write privileges for this file.
149	DFSFileDetached	The function was not performed because the file is detached.

Remarks

The **DFSExpandFile** function is required to expand a file or to write in a hole in a sparse file (**DFSWrite** and **DFSWriteNoWait** cannot expand a file by writing beyond the current end of the file or by writing in a hole in a sparse file. Also, normal (non-direct) writes which could normally expand a file are rejected by the OS while a file is in direct file mode).

Since it is always possible that DFS might find that some of the blocks in the indicated range have been allocated by other threads or processes after the caller determined that they were free, the caller must handle this contingency. It is logical that the caller repeat the sequence of freeing limbo blocks and attempting to expand several times before reducing the number of contiguous blocks requested and making multiple requests. For details on striping and other allocation details, see Impact of Striping. New file space allocated to fill a hole in a sparse file is zero-filled. Contiguous blocks added to the end of allocated file space are not zero-filled.

NOTE: A range of blocks that spans two volume segments is not

considered contiguous, even though the logical volume block addresses are contiguous.

See Also

DFSFreeLimboVolumeSpace

DFSFreeLimboVolumeSpace

Requests DFS to free a number of limbo blocks on a volume
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSFreeLimboVolumeSpace (
   LONG   volumeNumber,
   LONG   numberOfBlocks);
```

Parameters

volumeNumber

(IN) The volume number where the requested number of limbo blocks are to be freed.

numberOfBlocks

(IN) Specifies the number of limbo blocks requested to be freed.

Return Values

0	DFSNormalCompletion	The operation was completed as specified.
22	DFSInsufficientLimboFileSpace	The request could not be completed because there were not enough contiguous limbo blocks to complete the request successfully.
152	DFSInvalidVolume	The volume does not exist or is not mounted.

Remarks

This function requests the OS to free a number of limbo blocks on a given volume. This function performs the equivalent of a purge of one or more files until it has freed the requested number of blocks (or more). There is no guarantee that the OS can free as many blocks as requested by the caller, or that the blocks freed are contiguous. Also there is no way to

caller, or that the blocks freed are contiguous. Also there is no way to guarantee that the blocks will be made available on a specific volume segment.

Other processes, including system functions, can acquire blocks that have just been freed before they can be allocated. The OS normally stripes allocation of files when multiple segments exist for a volume, so it can be very difficult to find a large contiguous area of free blocks on a volume where non-direct or normal files are allocated in multisegment volumes. Callers should be prepared to call this function multiple times.

See Also

DFSExpandFile

DFSRead

Reads the sectors requested from a file using DFS (sleeps until completion)
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSRead (
   LONG   fileHandle,
   LONG   startingSector,
   LONG   sectorCount,
   BYTE   *buffer);
```

Parameters

fileHandle

(IN) The file handle returned from a prior call to **open** for the indicated file.

startingSector

(IN) The starting sector number (logical offset from beginning of the file) in the file where the read operation is to begin.

sectorCount

(IN) The number of sectors to be read into the buffer.

buffer

(OUT) Pointer to a contiguous buffer area large enough to contain the number of sectors indicated to be read.

Return Values

0	DFSNormalCompletion	The operation was completed as specified.
17	DFSHoleInFileError	A read was attempted to file sector addresses where no file blocks are allocated. The buffer is zero-filled.
18	DFSParameterError	The caller supplied one or more invalid parameters.

23	DFSNotInDirectFileMode	A direct file read (**DFSRead**) was issued but the file has not been opened successfully in direct mode.
24	DFSOperationBeyondEndOfFile	A read function was requested beyond the current end of file.
131	DFSHardIOError	
136	DFSInvalidFileHandle	A DFS call was made using a file handle which is not valid. Typically an open was omitted or the user inadvertently closed the file before attempting this access.
147	DFSNoReadPrivilege	Current connection does not have read privileges for the file.
149	DFSFileDetached	The function was not completed because the file is detached.
150	DFSInsufficientMemory	DFS could not allocate sufficient memory to complete the request.
162	DFSIOLockError	

Remarks

This function performs a read of one or more sectors using a logical zero-based sector offset into the indicated file. Since this function is blocking, control is returned to the caller after all reads relating to the requested read function have been completed. If a status indicating a hole was detected during the requested read operation, the buffer is zeroed. It is not possible to read beyond the end of the allocated area of a file.

See Also

DFSReadNoWait, **DFSWrite**

DFSReadNoWait

Reads the sectors requested from a file using DFS (returns after initiation)
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSReadNoWait (
   LONG      fileHandle,
   LONG      startingSector,
   LONG      sectorCount,
   BYTE      *buffer,
   struct DFSCallBackParameters
             *callBackNode);
```

Parameters

fileHandle

(IN) The file handle returned from a prior call to **open** for the indicated file.

startingSector

(IN) The starting sector number (logical offset from beginning of file) in the file where the read operation is to begin.

sectorCount

(IN) The number of sectors to be read into the buffer.

buffer

(OUT) Pointer to a contiguous buffer area large enough to contain the number of sectors indicated to be read.

callBackNode

(IN) Pointer to a structure used to signal completion of all requested reads for a particular call to **DFSReadNoWait**.

Return Values

0	Read Normal Initiation
!=0	Read not initiated

NOTE: The actual completion is stored in the completionCode field of the DFSCallBackParameters upon completion of the request.

Remarks

This function is identical to **DFSRead**, except that the return to the function caller is made immediately after posting the reads to the driver. This means that the status returned from the function only indicates whether the call was initiated or not. The completion status is returned in the structure provided for completion notification. A calling process must allow other processes to run. Consequently, any long sequence of code including this function call should make frequent calls to **ThreadSwitch** to allow other processes to be executed.

The DFSCallBackParameters structure is defined as follows:

```
struct DFSCallBackParameters
{
   LONG   localSemaphoreHandle;
   LONG   completionCode;
};
```

The *localSemaphoreHandle* field contains a local semaphore handle obtained by calling **OpenLocalSemaphore**. **WaitOnLocalSemaphore** or **ExamineLocalSemaphore** should be called to determine when the semaphore has been signalled.

The *completionCode* field contains the actual completion code, initialized to -1 by this function and updated upon completion. For completion code values, see **DFSRead**.

See Also

DFSRead, **DFSWriteNoWait**, **OpenLocalSemaphore**

DFSReturnFileMappingInformation

Returns file extents, each with the number of blocks, and starting file and volume block numbers

Local Servers: nonblocking

Remote Servers: N/A

Classification: 4.x

SMP Aware: No

Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSReturnFileMappingInformation (
   LONG   fileHandle,
   LONG   startingBlockNumber,
   LONG   *numberOfEntries,
   LONG   tableSize,
   struct FileMapStructure
          *table);
```

Parameters

fileHandle

(IN) Specifies the file handle returned from a prior call to **open** for the file.

startingBlockNumber

(IN) Specifies the starting file block address for which map is requested (zero relative).

numberOfEntries

(OUT) Receives the number of valid file map entries returned.

tableSize

(IN) Receives the number of file map entries for which space has been allocated by the caller in the following table (that is, max # of struct FileMapStructure to be returned).

table

(OUT) Receives a table of file map entries.

Return Values

0	DFSNormalCompletion	The operation was completed and information fields are valid.

18	DFSParameterError	The function caller supplied an invalid parameter.
136	DFSInvalidFileHandle	A DFS call was made using a file handle that is not valid. Typically an open was omitted or the user inadvertently closed the file before attempting this access.

Remarks

This function is required to provide the callingNLM™ application with details of exactly where a given file's logical blocks are located, including where file holes and the end of a file's allocated storage space is, so that the application can expand the file by calling **DFSExpandFile**.

Each file map entry has the following structure (found in nwdfs.h):

```
struct FileMapStructure
{
   LONG   fileBlock;
   LONG   volumeBlock;
   LONG   numberOfBlocks;
};
```

See FileMapStructure.

See Also

DFSReturnVolumeBlockInformation, **DFSReturnVolumeMappingInformation**

DFSReturnVolumeBlockInformation

Returns the volume block usage bitmap for requested volume
Local Servers: nonblocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSReturnVolumeBlockInformation (
   LONG   volumeNumber,
   LONG   startingBlockNumber,
   LONG   numberOfBlocks,
   BYTE   *buffer);
```

Parameters

volumeNumber

(IN) Specifies the volume number for which the volume block information is desired.

startingBlockNumber

(IN) Specifies volume logical block zero or an even multiple of 8 up to the last block of the volume.

numberOfBlocks

(IN/OUT) Specifies the number of blocks for which allocation bit flags are to be transferred into the buffer (If *startingBlockNumber* plus *numberOfBlocks* is greater than the total number of volume blocks, the number of volume blocks remaining starting from *startingBlockNumber* is substituted here).

buffer

(OUT) Receives a pointer to a buffer area where the information is returned. The area required for the buffer is the number of blocks rounded up modulus 8. The format of the data in the buffer is bit array, with 1 bits indicating available blocks. The relative bit address of each bit is the block address relative to the beginning of the specified starting file block number.

Return Values

0	DFSNormalCompl	The operation is complete and information

0	DFSNormalCompletion	The operation is complete and information fields are valid.
152	DFSInvalidVolume	The volume number specified does not exist or not mounted.

Remarks

This function is used to determine which blocks on a volume are in use and that are available for allocation. This function returns a bitmap which has a bit for each block in the range specified in the calling parameters, beginning with the logical (zero-based) volume block indicated by *startingBlockNumber*. This information is required if an application NLM is attempting to do specific allocation for a file, in order to pick block ranges of contiguous free blocks to expand a file.

The data returned by this function is only valid until it is changed by some request, and can change dynamically before an application can successfully request allocation of the blocks selected. The application process must be designed to handle this exception, as well as the case where there is not a single contiguous free block area large enough to satisfy the file expansion request.

See Also

DFSReturnFileMappingInformation, **DFSReturnVolumeMappingInformation**

DFSReturnVolumeMappingInformation

Returns information about a volume required for file allocation
Local Servers: nonblocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSReturnVolumeMappingInformation (
   LONG                    volumeNumber,
   struct VolumeInformationStructure
                           *volumeInformation);
```

Parameters

volumeNumber

(IN) Indicates the system volume number of the selected volume. A requesting process can determine the appropriate volume number using the File System Services functions.

volumeInformation

(OUT) Provides a structure of type VolumeInformationStructure (see below).

Return Values

0	DFSNormalCompletion	The operation is complete and information fields are valid.
152	DFSInvalidVolume	The volume number specified does not exist or is not mounted.

Remarks

This function provides volume information, including allocation block size, that is necessary to determine how many blocks to allocate in order to allocate a specified number of bytes for a request. This information changes dynamically. Therefore, it can become invalid before an application NLM can use the information. Application NLM applications must be designed to handle the likelihood that blocks available change

dynamically.

The VolumeInformationStructure structure defined as follows:

```
struct VolumeInformationStructure
{
   LONG   VolumeAllocationUnitSizeInBytes;
   LONG   VolumeSizeInAllocationUnits;
   LONG   VolumeSectorSize;
   LONG   AllocationUnitsUsed;
   LONG   AllocationUnitsFreelyAvailable;
   LONG   AllocationUnitsInDeletedFilesNotAvailable;
   LONG   AllocationUnitsInAvailableDeletedFiles;
   LONG   NumberOfPhysicalSegmentsInVolume;
   LONG   PhysicalSegmentSizeInAllocationUnits[64];
};
```

See VolumeInformationStructure.

See Also

DFSReturnFileMappingInformation,
DFSReturnVolumeBlockInformation

DFSSetEndOfFile

Sets file size
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSSetEndOfFile (
   LONG   fileHandle,
   LONG   newFileSize,
   LONG   returnTruncatedBlocksFlag);
```

Parameters

fileHandle

(IN) Specifies the file handle returned from a prior **DFSsopen** or **DFScreat** call for the indicated file.

newFileSize

(IN) Specifies the new file size in bytes.

returnTruncatedBlocksFlag

(IN) A nonzero value specified here indicates that any blocks truncated by the new file size should be freed up for future OS use.

Return Values

0	DFSNormalCompletion	The operation is complete as specified.
16	DFSTruncationFailure	The **DFSSetEndOfFile** function detected that the caller requested excess blocks to be truncated, but the file was open for other connections, causing the request to be rejected.
131	DFSHardIOError	
136	DFSInvalidFileHandle	A call was made with an invalid file handle. Typically an open was omitted or the user inadvertently closed the file.

Remarks

If the connection making the request is the only entity with the file open and if the *returnTruncatedBlocksFlag* is nonzero, setting a new file size that is one or more blocks less than the previous file size causes blocks (actually allocated) beyond the new defined file size to be truncated, or returned for future OS usage. If a new file size is specified that is greater than the previous file size, the newly defined file area is zero-filled, provided that actual file space is allocated. A new file size can be specified that is beyond the range of current allocated file space, or that is less than current allocated file space.

See Also

DFSExpandFile, **DFSReturnFileMappingInformation**, **DFSWrite**

DFSsopen

Opens the requested file in Direct File Mode
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSsopen (
   BYTE   *fileName,
   LONG   access,
   LONG   share,
   LONG   permission,
   LONG   flagBits,
   LONG   dataStream);
```

Parameters

fileName

(IN) Specifies the name of the file to be opened. The file name must be NULL-terminated and must include the path including the volume name but not the server name.

access

(IN) Specifies the access mode.

share

(IN) Specifies the sharing mode of the file.

permission

(IN) Specifies the access permissions for the file. The access permissions (see the sys\stat.h file) are as follows:

S_IWRITE	The file is writable
S_IREAD	The file is readable

flagBits

(IN) Specifies the following flags when a file is opened:

0x0000004	FILE_WRITE_THROUGH_BIT

0	
0x00010000	NO_RIGHTS_CHECK_ON_OPEN_BIT

dataStream

(IN) Specifies the name space.

Return Values

!=	DFSFailedCompletion	The requested operation is complete. The actual value returned is a file handle which is used for other functions that operate on the file.
-1	DFSFailedCompletion	The requested file was not opened.

If an error has occurred, *errno* can be set to

1	No such file
4	Bad file handle
6	Permission denied
9	Invalid argument

When an error occurs, *NetWareErrno* is set to

108	(0x6C)	ERR_BAD_ACCESS
152	(0x98)	ERR_INVALID_VOLUME
156	(0x9C)	ERR_INVALID_PATH

Remarks

The name of the file to be opened is given by the *filename* parameter. The file is accessed according to the access mode specified by the*access* parameter.

When a file is opened in direct file mode by calling **DFSsopen**, DFS flags

the file as being in direct file mode. In this mode, the cache and TTS are bypassed for future accesses to the file. Existing cache entries for the file are flushed and a turbo FAT for the file is built if one does not currently exist.

This could cause problems with other applications that have already opened the file in normal non-direct mode. In this case, the file is switched to direct mode, and the program with the file open in normal mode is able to read the file but cannot write to it. A close must be issued for each handle obtained by an open for the file before the file can be reopened for full normal mode access again (see Direct File System I/O).

The *access* parameter can have the following values as defined in FCNTL.H:

0x0000	O_RDONLY	open for read only
0x0001	O_WRONLY	open for write only
0x0002	O_RDWR	open for read and write
0x0010	O_APPEND	writes done at end of file
0x0020	O_CREAT	create new file
0x0040	O_TRUNC	truncate existing file
0x0080	O_EXCL	exclusive open

The *share* parameter can have the following values as defined in NWSHARE.H:

SH_COMPAT	Sets compatibility mode
SH_DENYRW	Prevents read or write access to the file
SH_DENYWR	Prevents write access of the file
SH_DENYRD	Prevents read access of the file
SH_DENYNO	Permits both read and write access to the file

The *dataStream* parameter can have the following values as defined in nwfile.h:

0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

See Also

DFSclose, DFScreat

DFSWrite

Writes sectors into a file using DFS (sleeps until completion)
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSWrite (
   LONG   fileHandle,
   LONG   startingSector,
   LONG   sectorCount,
   BYTE   *buffer);
```

Parameters

fileHandle

(IN) Specifies the file handle returned from a prior **DFSsopen** or **DFScreat** call for the indicated file.

startingSector

(IN) Specifies the starting sector number in the file (logical offset from beginning of the file) where the write operation is to begin.

sectorCount

(IN) Specifies the number of sectors to be written from the buffer.

buffer

(IN) Points to a contiguous buffer area large enough to contain the number of sectors to be written.

Return Values

0	DFSNormalCompletion	The operation was completed as specified.
4	DFSVolumeSegmentDeactivated	The volume segments on which the file is located have been deactivated by the operating system.
17	DFSHoleInFileError	An attempt was made to write to a file block address for which space has not been allocated (a hole in a

		sparse file). Space must be allocated by calling **DFSExpandFile** to fill holes in sparse files before the associated file block address can be written to successfully when using DFS.
18	DFSParameterError	The function caller supplied an invalid parameter.
23	DFSNotInDirectFileMode	A function requiring a file to be opened in direct mode (using **DFSsopen** or **DFScreat**) was requested but the file is not open in direct mode.
131	DFSHardIOError	
136	DFSInvalidFileHandle	(A DFS call was made using a file handle which is not valid. Typically an open was omitted or the user inadvertently closed the file before attempting this access).
148	DFSNoWritePrivilege	The current permissions that the file has been opened with do not allow writes to this file.
149	DFSFileDetached	
150	DFSInsufficientMemory	The Direct File System could not obtain memory necessary to complete the requested function.
162	DFSIOLockError	

Remarks

This function performs a write of one or more sectors using a logical zero-based sector offset into the indicated file. Since this function is blocking, control is returned to the caller after all writes relating to the requested write function are completed. If a status indicating a hole was detected during the requested read operation, the operation failed. It is not possible to write beyond the end of the allocated area of a file, or in holes where no blocks are allocated. This function sets a new file size if a write is issued to allocated file space beyond the current file size.

See Also

DFSExpandFile, **DFSWriteNoWait**

DFSWriteNoWait

Writes sectors into a file using DFS (returns immediately after initiation)
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: Direct File System

Syntax

```
#include <nwdfs.h>

LONG DFSWriteNoWait (
   LONG            fileHandle,
   LONG            startingSector,
   LONG            sectorCount,
   BYTE            *buffer,
   struct DFSCallBackParameters
                   *callBackNode);
```

Parameters

fileHandle

(IN) Specifies the file handle returned from a prior **DFSsopen** call for the file.

startingSector

(IN) Specifies the starting sector number in the file (logical offset from beginning of file) where the write operation is to begin.

sectorCount

(IN) Specifies the number of sectors to be written from the buffer.

buffer

(IN) Points to a contiguous buffer area large enough to contain the number of sectors to be written.

callBackNode

(IN) Points to a structure used to signal completion of all requested writes for a particular call to **DFSWriteNoWait**.

Return Values

0	Write operation initiated
-1	Bad file handle

This function can also return the return status codes found in the **DFSWrite** information above.

Remarks

Operation is identical to **DFSWrite** except that the current thread of execution is not blocked until the completion of the requested operation (**DFSWrite** calls **DFSWriteNoWait**, then waits for the completion to be signalled).

The DFSCallBackParameters structure is defined as follows:

```
struct DFSCallBackParameters
{
   LONG   localSemaphoreHandle;
   LONG   completionCode;
};
```

The *localSemaphoreHandle* field contains a local semaphore handle obtained by calling **OpenLocalSemaphore**. **WaitOnLocalSemaphore** or **ExamineLocalSemaphore** should be called to determine when the semaphore has been signalled.

The *completionCode* field contains a zero (or the value already in the field if it has not been zeroed out before **DFSWriteNoWait** is called) if a bad file handle is passed, and a -1 for all other completions.

See Also

DFSExpandFile, **DFSWrite**

File Service Group

Direct File System: Structures

DFSCallBackParameters

Used to signal completion of **DFSReadNoWait**
Service: Direct File System
Defined In: nwdfs.h

Structure

```
struct DFSCallBackParameters
{
   LONG   localSemaphoreHandle;
   LONG   completionCode;
};
```

Fields

localSemaphoreHandle

Contains a local semaphore handle obtained by calling **OpenLocalSemaphore**.

completionCode

Contains the completion code for **DFSReadNoWait**.

Remarks

See **DFSReadNoWait** for more information.

FileMapStructure

Service: Direct File System
Defined In: nwdfs.h

Structure

```
struct FileMapStructure
{
   LONG   fileBlock;
   LONG   volumeBlock;
   LONG   numberOfBlocks;
};
```

Fields

fileBlock

Contains the starting logical block (zero-based) of the file for this extent or group of contiguous blocks.

volumeBlock

Contains the actual starting logical volume block (zero-based) of the contiguous volume blocks assigned to the logical file blocks above.

numberOfBlocks

Contains the number of contiguous volume blocks that compose this extent (extent block length).

VolumeInformationStructure

Contains information about a NetWare volume
Service: Direct File System
Defined In: nwdfs.h

Structure

```
struct VolumeInformationStructure
{
   LONG   VolumeAllocationUnitSizeInBytes;
   LONG   VolumeSizeInAllocationUnits;
   LONG   VolumeSectorSize;
   LONG   AllocationUnitsUsed;
   LONG   AllocationUnitsFreelyAvailable;
   LONG   AllocationUnitsInDeletedFilesNotAvailable;
   LONG   AllocationUnitsInAvailableDeletedFiles;
   LONG   NumberOfPhysicalSegmentsInVolume;
   LONG   PhysicalSegmentSizeInAllocationUnits[64];
};
```

Fields

VolumeAllocationUnitSizeInBytes

Contains the number of bytes contained in a block allocated by the OS (this can be 4K, 8K, 16K, 32K, or 64K).

VolumeSizeInAllocationUnits

Contains the number of blocks of the size indicated in the parameter above that are contained in a volume (the volume total size can be calculated with these two parameters).

VolumeSectorSize

Contains the size of each sector on a volume (currently only a sector size of 512 bytes is supported by the OS).

AllocationUnitsUsed

Contains the number of blocks on a volume used with current non-deleted files.

AllocationUnitsFreelyAvailable

Contains the number of blocks currently available for file allocation.

AllocationUnitsInDeletedFilesNotAvailable

Contains the number of blocks on a volume which compose files that have been deleted but for which the necessary time has not yet elapsed before they can be purged or moved to the *AllocationUnitsFreelyAvailable* category.

AllocationUnitsInAvailableDeletedFiles

Contains the number of blocks which compose files deleted for which the required time has expired prior to being purged, but which have not yet been purged or moved to the *AllocationUnitsFreelyAvailable* category.

NumberOfPhysicalSegmentsInVolume

Contains the number of physical volume segments that are linked to form a volume.

PhysicalSegmentSizeInAllocationUnits

Contains an array that specifies the number of blocks in each volume segment, of which a maximum of 64 are allowed per volume. This also allows an application process to determine at what point in the logical volume block number a transition takes place from one volume segment to another. This information is needed by applications doing specific file allocation.

DOS Partition

DOS Partition: Guides

DOS Partition: Concept Guide

Additional Links

Parent Topic:

DOS Partition: Concepts

DOS Partition Access

The DOS Partition functions allow developers to access files that are in a disk's DOS partition. These functions should be used only when it is **absolutely** necessary to access a file in the DOS partition. Accessing files in the DOS partition is much slower than accessing files in the NetWare® partition of a disk, and adversely affect other aspects of the server's performance.

The **DOS partition** refers to the set of files accessible from the server PC booted under DOS. The set of files includes files on floppy disks, on DOS partitions on hard disks, and on any other DOS drive.

A DOS drive can include RAM disks and network drives. However, accessing RAM disks and network drives from NetWare is not recommended. Both RAM disks and NetWare 3.x and above use extended memory and temporarily switch to protected mode, causing a conflict. Accessing network drives uses the LAN board. Since the NetWare 3.x and above OS, in most configurations, also uses this board, two programs would access the same board. For many kinds of LAN boards, this causes a deadlock. Therefore, NLM™ applications that use the DOS partition should only reference files on floppy or hard disks.

The DOS partition functions have been made available primarily for the installation of new software from the DOS partition to the NetWare partition. If a file in the DOS partition is to be accessed more than once or twice, it should be moved into the NetWare partition.

Additional functions for accessing the DOS partition are available through the File System Services and Operating System I/O Services. You can call these functions on the DOS partition just as you would for the NetWare partition. The functions available for NetWare 3.x and above are **open**, **fopen**, and **freopen**. The same functions are available for NetWare 4.x and above along with these additional functions: **access**, **chmod**, **remove**, and **rename**.

DOSPresent should be called before using any of the DOS partition functions.

Parent Topic:

DOS Partition: Guides

File Service Group

DOS Partition Introduction

> **NOTE:** DOS Partition Services provide functions for NLM development.

The DOS Partition functions allow developers to access files that are in a disk's DOS partition. These functions should be used only when it is **absolutely** necessary to access a file in the DOS partition. Accessing files in the DOS partition is much slower than accessing files in the NetWare® partition of a disk, and adversely affect other aspects of the server's performance.

The **DOS partition** refers to the set of files accessible from the server PC booted under DOS. The set of files includes files on floppy disks, on DOS partitions on hard disks, and on any other DOS drive.

A DOS drive can include RAM disks and network drives. However, accessing RAM disks and network drives from NetWare is not recommended. Both RAM disks and NetWare 3.x and above use extended memory and temporarily switch to protected mode, causing a conflict. Accessing network drives uses the LAN board. Since the NetWare 3.x and above OS, in most configurations, also uses this board, two programs would access the same board. For many kinds of LAN boards, this causes a deadlock. Therefore, NLM™ applications that use the DOS partition should only reference files on floppy or hard disks.

The DOS partition functions have been made available primarily for the installation of new software from the DOS partition to the NetWare partition. If a file in the DOS partition is to be accessed more than once or twice, it should be moved into the NetWare partition.

Additional functions for accessing the DOS partition are available through the File System Services and Operating System I/O Services. You can call these functions on the DOS partition just as you would for the NetWare partition. The functions available for NetWare 3.x and above are **open**, **fopen**, and **freopen**. The same functions are available for NetWare 4.x and above along with these additional functions: **access**, **chmod**, **remove**, and **rename**.

DOSPresent should be called before using any of the DOS partition functions.

Parent Topic:

DOS Partition: Guides

Summary of DOS Partition Functions

Table auto. DOS Partition Functions

Function	Purpose

DOSClose	Closes a file in the DOS partition.
DOSCopy	Copies a file from the DOS partition to the NetWare® partition.
DOSCreate	Creates a file in the DOS partition of the disk.
DOSFindFirstFile	Initializes a search for files in the DOS partition.
DOSFindNextFile	Searches for files in the DOS partition.
DOSOpen	Opens a file in the DOS partition of the disk.
DOSPresent	Determines whether DOS is still present in memory.
DOSRead	Reads from a file in the DOS partition.
DOSsopen	Opens a DOS file for shared access.
DOSWrite	Writes to a file in the DOS partition.

Parent Topic:

DOS Partition: Guides

DOS Partition: Functions

DOSClose

Closes a file in the DOS partition
Local Servers: blocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSClose (
   int   handle);
```

Parameters

handle

(IN) Specifies the file handle obtained with a previous call to **DOSCreate** or **DOSOpen**.

Return Values

0	(0x0 0)	ESUCCESS
DOSCode		UNSUCCESSFUL

Remarks

The **DOSClose** function closes a file in the DOS partition.

See Also

DOSCreate, DOSOpen, DOSPresent

DOSCopy

Copies a file from the DOS partition to the NetWare® partition.
Local Servers: blocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSCopy (
   char   *NetWareFileName,
   char   *DOSFileName);
```

Parameters

NetWareFileName

(IN) Specifies the name of the file in the NetWare partition; a full NetWare pathname, including a volume name, is allowed.

DOSFileName

(IN) Specifies the name of the file that is to be copied from the DOS partition; any legal DOS pathname is allowed.

Return Values

0	(0x0 0)	ESUCCESS
DOSCode		UNSUCCESSFUL

Remarks

The **DOSCopy** function copies an existing file from the DOS partition to the NetWare partition. If a file with the same name already exists in the NetWare partition, the new file overwrites it.

See Also

DOSPresent

DOSCreate

Creates a file in the DOS partition
Local Servers: blocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSCreate (
   char   *fileName,
   int    *handle);
```

Parameters

fileName

(IN) Specifies the DOS filename of the file to be created (any legal DOS pathname is allowed).

handle

(OUT) Receives a file handle which provides access to the DOS file.

Return Values

0	(0x0 0)	ESUCCESS
DOSCode		UNSUCCESSFUL

Remarks

If the file does not exist, **DOSCreate** creates it. If the file does exist, it is truncated to zero bytes in length. The file is created with read/write access.

See Also

DOSOpen, **DOSPresent**

DOSFindFirstFile

Used to search for files in the DOS partition
Local Servers: blocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSFindFirstFile (
   char            *fileName,
   WORD            searchAttributes,
   struct find_t   *diskTransferAddress);
```

Parameters

fileName

(IN) Specifies the name of the file to be found in the DOS partition (a full pathname, including a drive letter and wildcards, is allowed).

searchAttributes

(IN) Determines the type of file for which the search is to be made.

diskTransferAddress

(OUT) Receives DOS information about the file.

Return Values

0	(0x0 0)	ESUCCESS
DOSCode		UNSUCCESSFUL

Remarks

The **DOSFindFirstFile** function finds the first file that matches the *fileName* and *searchAttributes* parameters. If wildcards are used in the *fileName*, **DOSFindNextFile** can be called to find other files that also match the *fileName* and *searchAttributes*.

The following attributes, defined in the nwdos.h header file, can also be

obtained in find_->attrib. The search attributes are: _A_NORMAL, _A_SUDIR, A_HIDDEN, and _A_SYSTEM.

```
_A_NORMAL   0x00   /* Normal file; read/write permitted */
_A_RDONLY   0x01   /* Read-only file */
_A_HIDDEN   0x02   /* Hidden file */
_A_SYSTEM   0x04   /* System file */
_A_VOLID    0x08   /* Volume ID entry */
_A_SUBDIR   0x10   /* Subdirectory */
_A_ARCH     0x20   /* Archive file */
```

The find_t structure is defined in the nwdos.h file and has the following format:

```
struct find_t
{
   char             reserved[21]; /* Reserved by DOS */
   char             attrib;       /* File's attributes */
   unsigned short   wr_time;      /* File's time stamp (DOS
                                     format) */
   unsigned short   wr_date;      /* File's date stamp (DOS
                                     format) */
   LONG             size;         /* File size in bytes */
   char             name[13];     /* Name of file */
};
```

See Also

DOSFindNextFile, **DOSPresent**, *DOS Technical Reference* manual

DOSFindNextFile

Used to search for files in the DOS partition
Local Servers: blocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSFindNextFile (
   struct find_t   *diskTransferAddress);
```

Parameters

diskTransferAddress

(IN/OUT) Obtained from a previous call to **DOSFindFirstFile**; it receives information about the file.

Return Values

0	(0x0 0)	ESUCCESS
DOSCode		UNSUCCESSFUL

Remarks

Consecutive calls to this function return information about all DOS files which match the *fileName* and *searchAttributes* parameters specified in the call to **DOSFindFirstFile**.

The find_t structure is defined in the nwdos.h file and has the following format:

```
struct find_t
{
   char             reserved[21]; /* Reserved by DOS */
   char             attrib;       /* File's attributes */
   unsigned short   wr_time;      /* File's time stamp (DOS
                                     format) */
   unsigned short   wr_date;      /* File's date stamp (DOS
```

```
                                        format) */
   LONG             size;          /* File size in bytes */
   char             name[13];      /* Name of file */
};
```

See Also

DOSFindFirstFile, DOSPresent

DOSMkdir

Creates a directory on the DOS partition of the NetWare server
Local Servers: blocking
Remote Servers: blocking
Classification: 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSMkdir (
   char   *dirName);
```

Parameters

dirName
(IN) Indicates the DOS directory name to be created.

Return Values

0	Success
DOS error code	Failure

DOSOpen

Opens a file in the DOS partition of the disk
Local Servers: blocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSOpen (
   char   *fileName,
   int    *handle);
```

Parameters

fileName

(IN) Specifies the DOS filename of the file to be opened (any legal DOS pathname is allowed).

handle

(OUT) Receives a file handle which provides access to the DOS file.

Return Values

0	(0x0 0)	ESUCCESS
DOSCode		UNSUCCESSFUL

Remarks

The file is opened with read/write access.

See Also

DOSCreate, **DOSPresent**

DOSPresent

Determines whether DOS is still present in memory
Local Servers: nonblocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSPresent (void);
```

Return Values

This function returns a value of 1 if DOS is still present in memory. Otherwise, it returns a value of 0.

Remarks

DOS must be present in memory for any of the other DOS functions to work.

The **REMOVE DOS** console command is used to remove DOS from memory.

DOSRead

Reads from a file in the DOS partition
Local Servers: blocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSRead (
   int    handle,
   LONG   fileOffset,
   char   *buffer,
   LONG   numberOfBytesToRead,
   LONG   *numberOfBytesRead);
```

Parameters

handle

(IN) Specifies the file handle obtained with a previous call to **DOSOpen** or **DOSCreate**.

fileOffset

(IN) Specifies the position in the file at which the first character is to be read.

buffer

(OUT) Receives the data read from the file.

numberOfBytesToRead

(IN) Specifies the number of bytes that are to be read from the file.

numberOfBytesRead

(OUT) Receives the number of bytes that are actually read from the file.

Return Values

0	(0x0 0)	ESUCCESS
DOSCode		UNSUCCESSFUL

Remarks

The **DOSRead** function reads from a file in the DOS partition.

See Also

DOSPresent, DOSWrite

DOSRemove

Removes a file from the DOS Partition of the NetWare server
Local Servers: blocking
Remote Servers: blocking
Classification: 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSRemove (
   const char   *fileName);
```

Parameters

fileName
(IN) Indicates the DOS file name to be removed.

Return Values

0	Success
DOS error code	Failure

DOSRmdir

Removes a directory from the DOS Partition of the NetWare server
Local Servers: blocking
Remote Servers: blocking
NetWare Servers: 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSRmdir (
   char   *dirName);
```

Parameters

dirName
(IN) Indicates the DOS directory name to be removed.

Return Values

0	Success
DOS error code	Failure

DOSsopen

Opens a DOS file for shared access
Local Servers: blocking
Remote Servers: N/A
Classification: 3.11, 3.12, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSsopen (
   const char   *filename,
   int          access,
   int          share,
   int          permission);
```

Parameters

access

(IN) Specifies the access mode of the file on open, as defined in FCNTL.H.

share

(IN) Specifies the sharing mode of the file on open as defined in NWSHARE.H (DOS 3.0 and above; must be 0 in DOS 2.x).

permission

(IN) File permissions if file is created (file does not exist, and O_CREAT is true).

Normal---0x00 Read/write
Read Only---0x01 Read only

Return Values

This function returns a file handle if successful. Otherwise, it returns:

-1	The NetWare or DOS error is in *NetWareErrno*. (The DOS and NetWare error codes do not overlap. If *NetWareErrno* = -1, DOS is not present.)

Remarks

This function is similar to the NetWare partition function **sopen**, except that it does not support O_APPEND and O_BINARY.

The following access modes are defined in FCNTL.H:

```
O_RDONLY   0x0000     /* Open for read only */
O_WRONLY   0x0001     /* Open for write only */
O_RDWR     0x0002     /* Open for read and write */
```

The following can be ORed with the above.

```
O_CREAT   0x0020    /* Create new file if file does not
                       exist */
O_TRUNC   0x0040    /* Truncate existing file */
```

The following share modes are defined in NWSHARE.H:

```
SH_COMPAT    0x00    /* Compatibility mode */
SH_DENYRW    0x10    /* Deny read/write mode */
SH_DENYWR    0x20    /* Deny write mode */
SH_DENYRD    0x30    /* Deny read mode */
SH_DENYNO    0x40    /* Deny none mode */
```

See Also

DOSOpen, sopen

DOSUnlink

Removes a file from the DOS Partition of the NetWare server
Local Servers: blocking
Remote Servers: blocking
Classification: 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSUnlink (
   const char   *fileName);
```

Parameters

fileName
(IN) Indicates the DOS file name to be removed.

Return Values

0	Success
DOS error code	Failure

DOSWrite

Writes to a file in the DOS partition
Local Servers: blocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: DOS Partition

Syntax

```
#include <nwdos.h>

int DOSWrite (
   int    handle,
   LONG   fileOffset,
   char   *buffer,
   LONG   numberOfBytesToWrite,
   LONG   *numberOfBytesWritten);
```

Parameters

handle

(IN) Specifies the file handle obtained with a previous call to **DOSOpen or DOSCreate**.

fileOffset

(IN) Specifies the position in the file at which the first character is written.

buffer

(IN) Specifies the data to be written to the file.

numberOfBytesToWrite

(IN) Specifies the number of bytes in the buffer that are to be written to the file.

numberOfBytesWritten

(OUT) Receives the number of bytes actually written to the file.

Return Values

0	(0x0 0)	ESUCCESS
DOSCode		UNSUCCESSFUL

Remarks

The **DOSWrite** function writes to a file in the DOS partition.

See Also

DOSPresent, DOSRead

File Service Group

DOS Partition: Structures

find_t

Contains DOS file information
Service: DOS Partition
Defined In: nwdos.h

Structure

```
struct find_t
{
   char             reserved[21];
   char             attrib;
   unsigned short   wr_time;
   unsigned short   wr_date;
   LONG             size;
   char             name[13];
};
```

Fields

reserved

Reserved by DOS.

attrib

Contains the file's attributes.

wr_time

Contains the file's time stamp in DOS format.

wr_date

Contains the file's date stamp in DOS format.

size

Contains the file's size in bytes.

name

Contains the anme of the file.

Extended Attribute

Extended Attribute: Guides

Extended Attribute: Task Guide

Accessing Extended Attributes

Accessing Attribute Selection by Name

Scanning for Extended Attributes

Closing an Extended Attribute

Additional Links

Extended Attribute: Functions

Extended Attribute: Structures

Parent Topic:

Extended Attribute: Guides

Extended Attribute: Concept Guide

Extended Attribute Introduction

Manipulating Extended Attribute Byte: Example

Extended Attribute Functions

Additional Links

Extended Attribute: Functions

Extended Attribute: Structures

Parent Topic:

Extended Attribute: Guides

Extended Attribute: Tasks

Accessing Attribute Selection by Name

To access an extended attribute by name, prepare the NWEA structure by calling **NWOpenEA**.

> **NOTE:** **NWOpenEA** doesn't actually open the extended attribute, but fills in the fields in the NWEA structure. The **NWOpenEA** function uses the directory handle/file path of the associated file, the name of the extended attribute, and the name space as arguments.

You can also call the **NWGetEAHandleStruct** function to obtain the NWEA structure if you know the extended attribute name. This function requires a valid NW_IDX structure. Call **NWGetDirectoryBase** or **NWNSGetMiscInfo** . (**NWOpenEA** performs this step for you.)

Parent Topic:

Extended Attribute: Guides

Accessing Extended Attributes

Use the following functions to read and write to the extended attributes of a file:

NWReadEA

NWWriteEA

Both functions open the extended attribute before proceeding, and both require a valid NWEA structure as input.

> **NOTE:** There are several ways to prepare this structure before passing it to either function. Refer to the structure information before proceeding.

Parent Topic:

Extended Attribute: Guides

Closing an Extended Attribute

1. **Complete the read or write operation on an extended attribute before closing an extended attribute.**

   Partial read or write operations aren't allowed. Any data past the end of the last read or write operation is lost when the file is closed.

2. **Call NWCloseEA to close the extended attribute directory entry after accessing a file's extended attributes.**

Parent Topic:

Extended Attribute: Guides

Scanning for Extended Attributes

1. **Call NWFindFirstEA and NWFindNextEA to scan the extended attributes of a file.**

   NWFindFirstEA initializes the scan operation and **NWFindNextEA** returns an NWEA structure for each extended attribute. The returned structure can be passed to **NWReadEA** or **NWWriteEA**.

   NWFindFirstEA takes an NW_IDXstructure as input. NW_IDX must identify the file associated with the extended attribute.

2. **To prepare NW_IDX, call the Name Space Services function NWGetDirectoryBase.**

Parent Topic:

Extended Attribute: Guides

Extended Attribute: Concepts

Extended Attribute Functions

Extended Attribute Services include the functions listed below.

Function	Comment
NWCloseEA	Closes the specified extended attribute.
NWFindFirstEA	Initializes the process of scanning extended attributes.
NWFindNextEA	Returns NWEA for accessing the next extended attribute.
NWGetEAHandleStruct	Prepares NWEA to be used by **NWReadEA** or **NWWriteEA**.
NWOpenEA	Opens the specified extended attribute.
NWReadEA	Reads the next block of data from the specified extended attribute.
NWWriteEA	Writes data to an extended attribute. If the extended attribute doesn't exist, this function attempts to create it.

Parent Topic:

Extended Attribute: Guides

Extended Attribute Introduction

A file's extended attributes are stored as fields in a separate directory entry. This entry is not accessible through conventional means (that is, directory handles and path specifications). Instead, the entry and its fields are referenced by a NetWare® Extended Attribute (NWEA) structure. The structure includes the following:

Connection ID

Read/write position

Extended attribute handle

Volume number

Directory entry

Key used

Key length

Key

The information in the NWEA structure is for internal use only. Allocate and maintain space for the structure, but don't modify its values.

Parent Topic:

Extended Attribute: Guides

Extended Attribute: Functions

NWCloseEA

Closes the specified Extended Attribute
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Extended Attribute

Syntax

```
#include <nwnamspc.h>
#include <nwea.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY(NWCCODE) NWCloseEA (
   NW_EA_HANDLE N_FAR  *EAHandle);
```

Pascal Syntax

```
#include <nwea.inc>

Function NWCloseEA
  (Var EAHandle : NW_EA_HANDLE
) : NWCCODE;
```

Parameters

EAHandle

(IN) Points to NW_EA_HANDLE.

Return Values

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

0x0000	SUCCESSFUL
0x89CF	INVALID_EA_HANDLE
0x89D3	EA_VOLUME_NOT_MOUNTED

Remarks

NWCloseEA must be called to save any changes made to an Extended Attribute. **NWCloseEA** must be called after a complete Read and/or Write cycle, not after each read or write function. (**NWCloseEA** should not be called after a find.)

NW_EA_HANDLE is referenced in all Extended Attribute functions. NW_EA_HANDLE is for internal use only; do not manipulate NW_EA_HANDLE in any way.

NCP Calls

0x2222 86 01 Close Extended Attribute Handle

NWFindFirstEA

Initializes the find-first/find-next Extended Attribute process
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Extended Attribute

Syntax

```
#include <nwnamspc.h>
#include <nwea.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY(NWCCODE) NWFindFirstEA (
   NWCONN_HANDLE            conn,
   NW_IDX N_FAR            *idxStruct,
   NW_EA_FF_STRUCT N_FAR   *ffStruct,
   NW_EA_HANDLE N_FAR      *EAHandle,
   pnstr8                   EAName);
```

Pascal Syntax

```
#include <nwea.inc>

Function NWFindFirstEA
  (conn : NWCONN_HANDLE;
   Var idxStruct : NW_IDX;
   Var ffStruct : NW_EA_FF_STRUCT;
   Var EAHandle : NW_EA_HANDLE;
   EAName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

idxStruct

(IN) Points to the NW_IDX structure containing the directory entry index.

ffStruct

(OUT) Points to the NW_EA_FF_STRUCT structure.

EAHandle

(OUT) Points to the NW_EA_HANDLE structure for the Extended Attribute.

EAName

(OUT) Points to the name of the Extended Attribute (optional).

Return Values

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

0x0000	SUCCESSFUL
0x0001	No EAs
0x8801	INVALID_CONNECTION
0x8836	INVALID_PARAMETER

Remarks

If any EAs exist for the associated file, **NWFindFirstEA** returns the NW_EA_HANDLE structure. If no EAs exist, **NWFindFirstEA** returns 1.

The NW_EA_HANDLE structure can call the **NWReadEA** and/or **NWWriteEA** function. Therefore, you do not need to call the **NWGetEAHandleStruct** function after **NWFindFirstEA** to initialize a Read or Write.

If you do call the **NWGetEAHandleStruct** function in preparation for writing, use the *EAName* parameter. When you copy by calling either **NWFindFirstEA** or the **NWFindNextEA** function, you must use the *EAName* parameter. If the *EAName* parameter is not needed, it can be passed NULL.

Information for the NW_IDX structure is obtained by calling the **NWNSGetMiscInfo** or **NWGetDirectoryBase** function. Functions use the NW_IDX structure to hold information concerning the name space and directory entry index of a file. This is how an application associates an Extended Attribute with a particular directory entry.

The NW_EA_FF_STRUCT structure is used internally by **NWFindFirstEA**.

The NW_EA_HANDLE and NW_EA_FF_STRUCT structures are for internal use only; do not manipulate these structures in any way.

NWFindFirstEA will return INVALID_PARAMETER if NULL is passed to either the *ffStruct* or *EAHandle* parameters.

NCP Calls

0x2222 86 04 Enumerate Extended Attribute

See Also

NWFindNextEA, **NWGetDirectoryBase**, **NWGetEAHandleStruct**, **NWNSGetMiscInfo**

NWFindNextEA

Returns the NW_EA_HANDLE structure for the next Extended Attribute
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Extended Attribute

Syntax

```
#include <nwnamspc.h>
#include <nwea.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY(NWCCODE) NWFindNextEA(
   NW_EA_FF_STRUCT N_FAR  *ffStruct,
   NW_EA_HANDLE N_FAR     *EAHandle,
   pnstr8                  EAName);
```

Pascal Syntax

```
#include <nwea.inc>

Function NWFindNextEA
  (Var ffStruct : NW_EA_FF_STRUCT;
   Var EAHandle : NW_EA_HANDLE;
   EAName : pnstr8
) : NWCCODE;
```

Parameters

ffStruct

(IN/OUT) Points to the NW_EA_FF_STRUCT structure returned by the **NWFindFirstEA** function.

EAHandle

(OUT) Points to the NW_EA_HANDLE structure.

EAName

(OUT) Points to the name of the Extended Attribute (optional).

Return Values

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

0x0000	SUCCESSFUL
0x0001	EA_DONE
0x8996	ERR_NO_ALLOC_SPACE
0x89C9	ERR_EA_NOT_FOUND
0x89CF	ERR_INVALID_EA_HANDLE
0x89D1	ERR_EA_ACCESS_DENIED
0x89FB	ERR_UNKNOWN_REQUEST
0x89FF	ERR_BAD_PARAMETER

Remarks

Before calling **NWFindNextEA**, you must call the **NWFindFirstEA** function. **NWFindNextEA** can then be called multiple times until all EAs have been found. EA_DONE is returned when there are no more EAs.

The NW_EA_HANDLE structure can also call the **NWReadEA** and/or **NWWriteEA** function. Therefore, do not call the **NWGetEAHandleStruct** function after the **NWFindFirstEA** function in order initialize a Read or Write.

If you do call the **NWGetEAHandleStruct** function in preparation for a Write, use the *EAName* parameter. When you copy by calling either the **NWFindFirstEA** or **NWFindNextEA** function, the *EAName* parameter must be used. If the *EAName* parameter is not needed, pass NULL.

The NW_EA_FF_STRUCT structure is used by the **NWFindFirstEA** function to return a handle to the first or next Extended Attribute.

The NW_EA_HANDLE and NW_EA_FF_STRUCT structures are for internal use only; do not manipulate these structures in any way.

NCP Calls

0x2222 86 04 Enumerate Extended Attribute

See Also

NWFindFirstEA

NWGetEAHandleStruct

Fills the NW_EA_HANDLE structure for use in the **NWReadEA** and **NWWriteEA** functions

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Extended Attribute

Syntax

```
#include <nwnamspc.h>
#include <nwea.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY(NWCCODE) NWGetEAHandleStruct(
   NWCONN_HANDLE         conn,
   pnstr8                EAName,
   NW_IDX N_FAR         *idxStruct,
   NW_EA_HANDLE N_FAR   *EAHandle);
```

Pascal Syntax

```
#include <nwea.inc>

Function NWGetEAHandleStruct
  (conn : NWCONN_HANDLE;
   EAName : pnstr8;
   Var idxStruct : NW_IDX;
   Var EAHandle : NW_EA_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

EAName

(IN) Points to the string containing the name of the Extended Attribute.

idxStruct

(IN) Points to the NW_IDX structure containing the directory entry index.

EAHandle

(IN/OUT) Points to the NW_EA_HANDLE structure containing the handle of the current Extended Attribute.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

The NW_EA_HANDLE structure is referenced in all Extended Attribute functions. The **NWReadEA** and **NWWriteEA** functions use the NW_EA_HANDLE structure to maintain state information. The state information is required to access the Extended Attribute database.

The NW_IDX structure information is obtained by calling the **NWNSGetMiscInfo** or **NWGetDirectoryBase** function. Functions use the NW_IDX structure to hold information about the name space and directory entry index of a file. This is how an application associates an Extended Attribute with a particular directory entry.

NCP Calls

None

See Also

NWFindFirstEA, **NWFindNextEA**, **NWGetDirectoryBase**, **NWNSGetMiscInfo**, **NWReadEA**, **NWWriteEA**

NWOpenEA

Fills the NW_EA_HANDLE structure so it can be used by the **NWReadEA** and **NWWriteEA** functions

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Extended Attribute

Syntax

```
#include <nwnamspc.h>
#include <nwea.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY(NWCCODE) NWOpenEA (
   NWCONN_HANDLE         conn,
   NWDIR_HANDLE          dirHandle,
   pnstr8                path,
   pnstr8                EAName,
   nuint8                nameSpace,
   NW_EA_HANDLE N_FAR   *EAHandle);
```

Pascal Syntax

```
#include <nwea.inc>

Function NWOpenEA
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   EAName : pnstr8;
   nameSpace : nuint8;
   Var EAHandle : NW_EA_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle pointing to the directory to search.

path

(IN) Points to a path.

EAName

(IN) Points to the string containing the name of the Extended Attribute.

nameSpace

(IN) Specifies the name space of the Extended Attribute.

EAHandle

(IN/OUT) Points to the NW_EA_HANDLE structure containing the handle of the current Extended Attribute.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

NWOpenEA combines the functionality of the **NWGetDirectoryBase** and **NWGetEAHandleStruct** functions in one function.

The **NWFindFirstEA** and **NWFindNextEA** functions also return a filled NW_EA_HANDLE structure.

NCP Calls

0x2222 87 22 Generate Directory Base and Volume Number

See Also

NWFindFirstEA, NWFindNextEA, NWGetDirectoryBase, NWGetEAHandleStruct, NWNSGetMiscInfo, NWReadEA, NWWriteEA

NWReadEA

Reads the next block of data from the specified Extended Attribute
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Extended Attribute

Syntax

```
#include <nwnamspc.h>
#include <nwea.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY(NWCCODE) NWReadEA (
   NW_EA_HANDLE N_FAR  *EAHandle,
   nuint32              bufferSize,
   pnuint8              buffer,
   pnuint32             totalEASize,
   pnuint32             amountRead);
```

Pascal Syntax

```
#include <nwea.inc>

Function NWReadEA
  (Var EAHandle : NW_EA_HANDLE;
   bufferSize : nuint32;
   buffer : pnuint8;
   totalEASize : pnuint32;
   amountRead : pnuint32
) : NWCCODE;
```

Parameters

EAHandle

(IN/OUT) Points to the NW_EA_HANDLE structure, obtained by calling either the **NWGetHandleStruct**, **NWFindFirstEA**, **NWFindNextEA**, or **NWOpenEA** function.

bufferSize

(IN) Specifies the size of the buffer.

buffer

(OUT) Points to a data buffer.

totalEASize

(OUT) Points to the size of the Extended Attribute.

amountRead

(OUT) Points to the total amount of data read with the call (not cumulative across multiple calls).

Return Values

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

0x0000	SUCCESSFUL
0x8833	INVALID_BUFFER_LENGTH
0x8988	INVALID_FILE_HANDLE
0x898C	NO_MODIFY_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x899C	INVALID_PATH
0x89C9	EA_NOT_FOUND
0x89CE	EA_BAD_DIR_NUM
0x89CF	INVALID_EA_HANDLE
0x89D1	EA_ACCESS_DENIED
0x89D3	EA_VOLUME_NOT_MOUNTED
0x89D5	INSPECT_FAILURE

Remarks

The data block to be read is determined from the state information identified by the *EAHandle* parameter.

NWReadEA and the **NWWriteEA** function can perform multiple actions, such as opening or creating an Extended Attribute and then calling the appropriate function. To properly end **NWReadEA**, call the **NWCloseEA** function after the last Read or Write.

Extended Attribute values should always be read or written completely. Extended Attributes are not treated like files when transferring. Therefore, partial Reads or Writes are not allowed.

If 0x0000 is returned, more data can be written to the Extended Attribute. If 0x0001 is returned, the data in the buffer was written successfully; however, no more data can be written to the Extended Attribute. If other errors are returned, the data in the buffer was not written to the Extended Attribute.

CAUTION: If an Extended Attribute is not read or written completely, data past the end of the last Read or Write may be lost!

The NW_EA_HANDLE structure is referenced in all Extended Attribute functions. **NWReadEA** and the **NWWriteEA** function use the NW_EA_HANDLE structure to maintain state information. The state information is required to access the Extended Attribute database. The NW_EA_HANDLE structure is for internal use only; do not manipulate it in any way.

Before calling **NWReadEA** initially, you must obtain the *EAHandle* parameter to access the Extended Attribute database. An application can obtain an Extended Attribute handle by calling one of the following functions:

NWFindFirstEA
NWFindNextEA
NWGetEAHandleStruct
NWOpenEA

NWReadEA can be called multiple times until the bytes of data read is equal to the value identified by the *totalEASize* parameter.

NOTE: The value referenced by the *amountRead* parameter does not reflect the total number of bytes in the Extended Attribute.

For Reads, the *bufferSize* parameter must be at least 512 bytes; it can be greater than 512 bytes---but must be in multiples of 512. If the *bufferSize* parameter is less than 512 bytes, **NWReadEA** returns INVALID_BUFFER_LENGTH.

The **NWEARead** function reads up to thenumber of bytes specified by the *bufferSize* parameter or until the end of the Extended Attribute data, whichever comes first.

NCP Calls

0x2222 86 03 Read Extended Attribute

See Also

NWFindNextEA, **NWFindFirstEA**, **NWOpenEA**, **NWWriteEA**

NWWriteEA

Writes data to an Extended Attribute
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Extended Attribute

Syntax

```
#include <nwnamspc.h>
#include <nwea.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY(NWCCODE) NWWriteEA (
   NW_EA_HANDLE N_FAR  *EAHandle,
   nuint32              totalWriteSize,
   nuint32              bufferSize,
   pnuint8              buffer,
   pnuint32             amountWritten);
```

Pascal Syntax

```
#include <nwea.inc>

Function NWWriteEA
  (Var EAHandle : NW_EA_HANDLE;
   totalWriteSize : nuint32;
   bufferSize : nuint32;
   buffer : pnuint8;
   amountWritten : pnuint32
) : NWCCODE;
```

Parameters

EAHandle

(IN/OUT) Points to the NW_EA_HANDLE structure returned by the **NWGetEAHandleStruct**, **NWFindFirstEA**, **NWFindNextEA**, or **NWOpenEA** function.

totalWriteSize

(IN) Specifies the size of the total Write.

bufferSize

(IN) Specifies the size of the buffer.

buffer

(IN) Points to a data buffer.

amountWritten

(OUT) Points to the amount of data written by **NWWriteEA** (not cumulative across multiple calls).

Return Values

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

0x0000	SUCCESSFUL; valid data remains in the Extended Attribute
0x0001	SUCCESSFUL; valid data remains in the buffer, not the Extended Attribute
0x8901	ERR_INSUFFICIENT_SPACE
0x898C	NO_MODIFY_PRIVILEGES
0x899C	INVALID_PATH
0x89C8	MISSING_EA_KEY
0x89C9	EA_NOT_FOUND
0x89CB	EA_NO_KEY_NO_DATA
0x89CE	EA_BAD_DIR_NUM
0x89CF	INVALID_EA_HANDLE
0x89D0	EA_POSITION_OUT_OF_RANGE
0x89D1	EA_ACCESS_DENIED
0x89D2	DATA_PAGE_ODD_SIZE
0x89D3	EA_VOLUME_NOT_MOUNTED
0x89D4	BAD_PAGE_BOUNDARY
0x89FF	HARDWARE_FAILURE

Remarks

If the Extended Attribute does not exist, **NWWriteEA** attempts to create it.

NWWriteEA returns 0x0000 when there is more data in the Extended Attribute and **NWWriteEA** needs to be called again. **NWWriteEA** returns 0x0001 when there is valid data in the buffer but none left in the Extended Attribute.

The **NWReadEA** function and **NWWriteEA** can perform multiple actions, such as opening or creating an Extended Attribute and then performing

the appropriate function. To properly end **NWWriteEA**, the **NWCloseEA** function must be called after the last Read, Write, and/or Find Extended Attribute function.

Extended Attribute values should always be read or written completely. Extended Attributes are not treated like files when transferring. Therefore, partial Reads or Writes are not allowed.

CAUTION: If an Extended Attribute is not read or written completely, data past the end of the last Read or Write may be lost!

Before calling **NWWriteEA**, an application must properly initialize the NW_EA_HANDLE structure to access the Extended Attribute database. An application can initialize the NW_EA_HANDLE structure by calling the **NWFindFirstEA**, **NWFindNextEA**, **NWGetEAHandleStruct**, or **NWOpenEA** function. The NW_EA_HANDLE structure is for internal use only; do not manipulate it in any way.

For Writes, the *bufferSize* parameter should be at least 512 bytes. If the *bufferSize* parameter is less than the *totalWriteSize* parameter, it must be a multiple of 512.

NWWriteEA writes up to the number of bytes specified by the *bufferSize* parameter or until the end of the Extended Attribute data, whichever comes first. If the data to be written is larger than the buffer size, **NWWriteEA** must be called multiple times to write all the data to the Extended Attribute.

An application should complete the entire Write before closing the Extended Attribute.

NCP Calls

0x2222 86 02 Write Extended Attribute

See Also

NWReadEA

Extended Attribute: Structures

NW_EA_FF_STRUCT

Maintains state information when scanning an extended attribute file
Service: Extended Attribute
Defined In: nwea.h

Structure

```
typedef struct
{
   NWCONN_HANDLE   connID;
   nuint16         nextKeyOffset;
   nuint16         nextKey;
   nuint32         numKeysRead;
   nuint32         totalKeys;
   nuint32         EAHandle;
   nuint16         sequence;
   nuint16         numKeysInBuffer;
   nuint8          enumBuffer[512];
} NW_EA_FF_STRUCT;
```

Pascal Structure

```
Defined in nwea.inc

 NW_EA_FF_STRUCT = Record
    connID : NWCONN_HANDLE;
    nextKeyOffset : nuint16;
    nextKey : nuint16;
    numKeysRead : nuint32;
    totalKeys : nuint32;
    EAHandle : nuint32;
    sequence : nuint16;
    numKeysInBuffer : nuint16;
    enumBuffer : Array[0..511] Of nuint8
  End;
```

Fields

connID

Specifies a connection to the server storing the Extended Attribute.

nextKeyOffset

Specifies a value that the server uses as part of an internal handle.

nextKey

Specifies a value that the server uses as part of an internal handle.

numKeysRead

Specifies the number of keys that have been read from the extended attribute file.

totalKeys

Specifies the total number of keys in the extended attribute file.

EAHandle

Specifies the handle for the current key.

sequence

Specifies the current key number.

numKeysInBuffer

Specifies the number of keys in the current buffer.

enumBuffer

Specifies the current buffer containing keys read from the extended attribute file.

Remarks

NW_EA_FF_STRUCT is an internal handle for library use only. Appplications must not modify this structure in any way.

NW_EA_HANDLE

Defines information associated with the extended attribute handle
Service: Extended Attribute
Defined In: nwea.h

Structure

```
typedef struct
{
   NWCONN_HANDLE   connID;
   nuint32         rwPosition;
   nuint32         EAHandle;
   nuint32         volNumber;
   nuint32         dirBase;
   nuint8          keyUsed;
   nuint16         keyLength;
   nuint8          key[256];
} NW_EA_HANDLE;
```

Pascal Structure

```
Defined in nwea.inc

  NW_EA_HANDLE = Record
    connID : NWCONN_HANDLE;
    rwPosition : nuint32;
    EAHandle : nuint32;
    volNumber : nuint32;
    dirBase : nuint32;
    keyUsed : nuint8;
    keyLength : nuint16;
    key : Array[0..255] Of nuint8
  End;
```

Fields

connID

Specifies the server storing the Extended Attribute.

rwPosition

Specifies the current position within the Extended Attribute file.

EAHandle

Specifies the handle to the Extended Attribute file.

volNumber

Specifies the volume storing the Extended Attribute file.

dirBase

Specifies the directory base associated with the Extended Attribute file.

keyUsed

Specifies the key used to access the Extended Attribute.

keyLength

Specifies the length of the *key* parameter.

key

Specifies the Extended Attribute key.

Remarks

NW_EA_HANDLE is an internal handle for library use only. Appplications must not modify this structure in any way.

T_enumerateEAnoKey

Describes the data layout returned by the **EnumerateEA** function when a key value is not specified

Service: Extended Attribute

Defined In: nwextatt.h

Structure

```
typedef struct
{
   LONG   valueLength;
   WORD   keyLength;
   LONG   accessFlags;
   char   keyValue[1];
} T_enumerateEAnoKey;
```

Fields

valueLength

Specifies the length of the Extended Attribute corresponding to the key.

keyLength

Specifies the length of the key value, which starts at the *keyValue* field.

accessFlags

Specifies developer-defined access flags.

keyValue

Specifies the first character of the key value.

Remarks

The key is a developer-defined value used for catagorizing Extended Attributes.

T_enumerateEAwithKey

Describes the data layout returned by the **EnumerateEA** function when a key value is specified

Service: Extended Attribute

Defined In: nwextatt.h

Structure

```
typedef struct
{
   LONG   EALength;
   WORD   keyLength;
   LONG   accessFlags;
   LONG   keyExtants;
   LONG   valueExtants;
   char   keyValue[1];
} T_enumerateEAwithKey;
```

Fields

EALength

Specifies the length of the Extended Attribute.

keyLength

Specifies the length of the *keyValue* field.

accessFlags

Specifies developer-defined access flags.

keyExtants

Specifies the number of 128-byte extants used by the key value.

valueExtants

Specifies the number of 128-byte extants used by the Extended Attribute.

keyValue

Specifies the first character of the key value.

Remarks

The key is a developer-defined value used for catagorizing Extended Attributes.

File Engine

File Engine: Functions

CountComponents

Returns the number of components contained in a NetWare® pathname
Local Servers: blocking
Remote Servers: blocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: Yes
Service: File Engine

Syntax

```
#include <nwfile.h>

int CountComponents (
   BYTE   *pathString,
   int    len);
```

Parameters

pathString

(IN) Specifies the string containing the NetWare pathname.

len

(IN) Specifies the length (in bytes) of the *pathString*.

Return Values

This function returns the number of components in *pathString*.

Remarks

This function works only with NetWare pathnames, which can consist of a directory path, filename, and filename extension.

A NetWare path consists of a path string and a path count. The path string does not use any type of delimiter character between components of the path. Instead, the length of each path component is specified in the byte immediately preceding each component of the path string. The path count tells how many path components there are in a path. This is the number returned by **CountComponents**.

For example, a normal path might look like this:

```
serverName/vol2:first/second/third/file.dat
```

If serverName is assigned file server ID 1, and vol2 is assigned volume

number 2, then the corresponding NetWare path format would be:

```
fileServerID = 1  volumeNumber = 2  pathString = 5first6second5third8fi
```

The *fileServerID* and *volumeNumber* are not actually part of the *pathString*, but are kept as separate numeric values. The numbers that are part of the *pathString* are actual binary values, not their ASCII equivalents. The *pathString* is the entity that would be passed to **CountComponents** (with a length of 28, which is the total length of *pathString*), and the returned component count would be 4 (the number of component parts in *pathString*).

See Also

_makepath, **_splitpath**

FEConvertDirectoryNumber

Converts a directory number in one name space to the comparable directory number in another name space

Local Servers: blocking

Remote Servers: N/A

Classification: 3.x, 4.x

Platform: NLM

SMP Aware: No

Service: File Engine

Syntax

```
#include <nwfile.h>

int FEConvertDirectoryNumber (
   int    sourceNameSpace,
   LONG   volumeNumber,
   LONG   sourceDirectoryNumber,
   int    destinationNameSpace,
   LONG   *destinationDirectoryNumberP);
```

Parameters

sourceNameSpace

(IN) Specifies the name space of the directory number to be converted.

volumeNumber

(IN) Specifies the volume number of the directory number to be converted.

sourceDirectoryNumber

(IN) Specifies the directory number that is to be converted.

destinationNameSpace

(IN) Specifies the name space to which the directory number is to be converted.

destinationDirectoryNumberP

(OUT) Receives the converted directory number which corresponds to the destination name space.

Return Values

This function returns a value of 0 if successful. Otherwise, it returns a nonzero value.

Remarks

A single directory entry has a different directory number for each name space that is supported on a volume. This function converts a directory number in one name space to the comparable directory number in another name space.

See Also

FEMapHandleToVolumeAndDirectory, **FEMapPathVolumeDirToVolumeDir**

FEcreat

Creates a file
Local Servers: blocking
Remote Servers: blocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

int FEcreat (
   char   *name,
   int    permission,
   int    flagBits);
```

Parameters

name

(IN) Specifies the name of the file to be opened.

permission

(IN) Specifies the file permission (if the file is being created).

flagBits

(IN) Specifies the special flags that allow more file flexibility.

Return Values

When there is no error opening the file, the function returns a file handle. When an error occurs, it returns a value of -1, and *errno* and *NetWareErrno* are set to the appropriate error codes.

Remarks

This function also works on the DOS partition.

This is a special version of **creat**.

If the specified file does not exist, **FEcreat** creates the file with the specified file permission.

The permission mode is established as a combination of bits found in the SYS\STAT.H file. The following bits are defined:

S_IWRITE	The file is writeable.
S_IREAD	The file is readable.

A value of 0 can be specified to indicate that the file is readable and writeable.

The flag bits can be found in the nwfile.h file and are defined as follows:

DELETE_FILE_ON_CREATE_BIT	If the file already exists, it is deleted. This allows the file to be created again.
NO_RIGHTS_CHECK_ON_OPEN_BIT	The user's rights to the file are not checked when the file is opened.
NO_RIGHTS_CHECK_ON_CREATE_BIT	The user's rights to the file are not checked when the file is created.
FILE_WRITE_THROUGH_BIT	When a file write is performed, the write function does not return until the data is actually written to the disk.
ENABLE_IO_ON_COMPRESSED_DATA_BIT	Any subsequent I/O on this entry is compressed (NetWare 4.x)
LEAVE_FILE_COMPRESSED_DATA_BIT	After all I/O has been done, leave this file compressed (NetWare 4.x)

See Also

close

FEFlushWrite

Flushes all pending writes for a file
Local Servers: blocking
Remote Servers: N/A
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

int FEFlushWrite (
   int   handle);
```

Parameters

handle

(IN) Specifies handle of the file to be flushed.

Return Values

This function returns a value of 0 if successful. Otherwise, it returns a NetWare error code.

Remarks

When this function returns, all writes associated with the file specified by the file handle are complete.

FEGetCWDnum

Returns the current working directory (CWD) number
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

LONG FEGetCWDnum (void);
```

Return Values

This function returns the CWD number (the default directory) for the current thread group.

Remarks

This function can be used by a registered path parsing function to get the CWD number when the path being parsed is a relative path.

See Also

FESetCWDnum, FESetCWVandCWDnums, FESetCWVnum

FEGetCWVnum

Returns the current working volume (CWV) number
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

LONG FEGetCWVnum (void);
```

Return Values

This function returns the CWV number (the default volume) for the current thread group.

Remarks

This function can be used by a registered path parsing function to get the CWV number when the path being parsed does not include a volume name.

See Also

FEGetCWDnum, **FESetCWDnum**, **FESetCWVandCWDnums**, **FESetCWVnum**

FEGetEntryVersion

Returns the version number for a directory entry (files or directories)
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

LONG FEGetEntryVersion (
   LONG   volumeNumber,
   LONG   directoryNumber,
   BYTE   *pathString,
   LONG   pathCount,
   WORD   *version);
```

Parameters

volumeNumber

(IN) Specifies the volume number on which the entry is located.

directoryNumber

(IN) Specifies the directory number used by the directory entry.

pathString

(IN) Points to a NetWare style path string relative to the volume/directory number. This is the name of the directory entry.

pathCount

(IN) Specifies the number of elements in the path string.

version

(OUT) Specifies the version number for the entry.

Return Values

0	(0x00)	Success
255	(0xFF)	Failure

Remarks

This function returns the version number for a specified directory entry. The version number of a directory entry is incremented once each time the entry is modified.

See Also

readdir, stat (Function)

FEGetOpenFileInfo

Returns directory entry information for a given connections file handle
Local Servers: blocking
Remote Servers: blocking
Classification: 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

int FEGetOpenFileInfo (
   LONG   connection,
   LONG   handle,
   LONG   *volume,
   LONG   *directoryNumber,
   LONG   *dataStream
   LONG   *flags);
```

Parameters

connection

(IN) Indicates the connection number of the object that has the file open.

handle

(IN) Indicates the file handle for which to return *volume* or *directoryNumber*.

volume

(OUT) Points to the number of the volume on which the directory entry is located.

directoryNumber

(OUT) Points to the directory entry number of the entry.

dataStream

(OUT) Points to the data stream with which the handle is associated.

flags

(OUT) Points to the status of the handle.

Return Values

0	Success
0xFF	Failure

Remarks

When given a connection number and a NetWare file handle, **FEGetOpenFileInfo** returns the information in the output parameters. The file handle for the *handle* parameter must be an OS file handle such as the *fileHandle* field returned in various FS Hooks return structures defined in nwfshook.h.

FEGetOpenFileInfo is useful if you are using FS Hooks because it gives the status/flags for an open file. However, keep in mind that *fileHandle* may not be populated by some callbacks---for example FSHOOK_PRE_OPENFILE if the file has not yet been opened. Also keep in mind that **FEGetOpenFileInfo** is a blocking function and cannot be used in a POST FS Hooks routine. In that case callback information would have to be passed to another routine to call **FEGetOpenFileInfo**.

FEGetOpenFileInfoForNS

Returns name space specific directory entry information for a given connections file handle

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x

Platform: NLM

SMP Aware: No

Service: File Engine

Syntax

```
#include <nwfile.h>

int FEGetOpenFileInfoForNS (
   LONG   connection,
   LONG   handle,
   LONG   *volume,
   LONG   *DOSDirectoryNumber,
   LONG   *directoryNumber,
   LONG   *nameSpace,
   LONG   *dataStream
   LONG   *flags);
```

Parameters

connection

(IN) Indicates the connection number of the object that has the file open.

handle

(IN) Indicates the file handle for which to return *volume* or *directoryNumber*.

volume

(OUT) Points to the number of the volume on which the directory entry is located.

DOSDirectoryNumber

(OUT) Points to the DOS directory entry number of the entry.

directoryNumber

(OUT) Points to the directory entry number of the entry corresponding with *nameSpace*.

nameSpace

(OUT) Points to the name space corresponding with *directoryNumber*.

dataStream

(OUT) Points to the data stream with which the handle is associated.

flags

(OUT) Points to the status of the handle.

Return Values

0	Success
0xFF	Failure

Remarks

When given a connection number and a NetWare file handle, **FEGetOpenFileInfoForNS** returns the information in the output parameters.

FEGetOpenFileInfo is useful if you are using FS Hooks because it gives the status/flags for an open file as well as some name space specific directory entry information.

FEGetOriginatingNameSpace

Gets the originating name space for a volume and directory number pair
Local Servers: blocking
Remote Servers: N/A
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

LONG FEGetOriginatingNameSpace (
   LONG   volumeNumber,
   LONG   directoryNumber);
```

Parameters

volumeNumber

(IN) Specifies the volume number for which the originating name space is desired.

directoryNumber

(IN) Specifies the directory number for which the originating name space is desired.

Return Values

This function returns a number indicating the originating name space for the volume and directory number pair, if successful. Otherwise, it returns a value of - 1, and *errno* and *NetWareErrno* contain appropriate error codes.

Remarks

This function provides useful information for file backup operations. With NetWare support for name spaces, knowing which name space created the file helps you determine the correct set of information to back up.

FEGetOriginatingNameSpace returns one of the following name spaces:

0	DOS
1	MACINTOSH

1	MACINTOSH
2	NFS*
3	FTAM
4	OS2 (OS/2*)
5	NT (MS Windows NT*)

See Also

SetCurrentNameSpace

FEMapConnsHandleToVolAndDir

Returns a volume number and a directory number for a given connection's file handle

Local Servers: blocking

Remote Servers: N/A

Classification: 3.12, 4.x

Platform: NLM

SMP Aware: No

Service: File Engine

Syntax

```
#include <nwfile.h>

int FEMapConnsHandleToVolAndDir (
   LONG   connectionNumber,
   int    handle,
   int    *volumeNumber,
   LONG   *directoryNumber);
```

Parameters

connectionNumber

(IN) Specifies the connection number of the object that owns the file handle.

handle

(IN) Specifies the file handle for which to return the volume and directory numbers.

volumeNumber

(OUT) Points to the number of the volume on which the directory entry is located.

directoryNumber

(OUT) Points to the directory entry number of the entry.

Return Values

0	(0x00)	Success.
255	(0xFF)	Failure

Other NetWare errors can be returned upon failure.

Remarks

When given a connection number and a file handle, this function returns a volume number and a directory number. This information can be used to get other information about the directory entry. The file handle can be obtained from normal CLIB file I/O or from the NetWare OS.

See Also

FEMapHandleToVolumeAndDirectory, **FEMapVolumeAndDirectoryToPath**

FEMapHandleToVolumeAndDirectory

Gets the volume and directory numbers being used by a file handle
Local Servers: blocking
Remote Servers: N/A
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

int FEMapHandleToVolumeAndDirectory (
   int    handle,
   int    *volumeNumberP,
   LONG   *directoryNumberP);
```

Parameters

handle

(IN) Specifies the file handle to be used to get the volume and directory numbers.

volumeNumberP

(OUT) Specifies the volume number used by the file handle.

directoryNumberP

(OUT) Specifies the directory number used by the file handle.

Return Values

This function returns a value of 0 if successful. Otherwise, it returns a NetWare error code.

Remarks

FEMapHandleToVolumeAndDirectory returns the volume and directory numbers used by the file handle.

See Also

FEMapPathVolumeDirToVolumeDir, **FEMapVolumeAndDirectoryToPath**, **FEMapVolumeNumberToName**

FEMapPathVolumeDirToVolumeDir

Maps a path consisting of a volume number, directory number, and pathname to a path consisting of a volume number and directory number

Local Servers: blocking

Remote Servers: N/A

Classification: 3.x, 4.x

Platform: NLM

SMP Aware: No

Service: File Engine

Syntax

```
#include <nwfile.h>

int FEMapPathVolumeDirToVolumeDir (
   char   *pathName,
   int    volumeNumber,
   LONG   directoryNumber,
   int    *newVolumeNumberP,
   LONG   *newDirectoryNumberP);
```

Parameters

pathName

(IN) Specifies the pathname for which the volume and directory number are desired.

volumeNumber

(IN) Specifies the volume number on which the pathname is based.

directoryNumber

(IN) Specifies the directory number on which the pathname is based.

newVolumeNumberP

(OUT) Receives the returned volume number.

newDirectoryNumberP

(OUT) Receives the returned directory number.

Return Values

This function returns a value of 0 if successful. Otherwise, it returns a NetWare error code.

Remarks

If the *pathName* parameter is a full volume pathname, a new volume and directory number are returned. If the path does not include a volume, *volumeNumber* is returned for *newVolumeNumberP*. If the path is relative, *newDirectoryNumberP* is based on the directory number and pathname.

See Also

FEMapHandleToVolumeAndDirectory, **FEMapPathVolumeDirToVolumeDir**, **FEMapVolumeNumberToName**

FEMapVolumeAndDirectoryToPath

Maps a volume number and directory number to a NetWare style path
Local Servers: blocking
Remote Servers: N/A
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

int FEMapVolumeAndDirectoryToPath (
   int    volumeNumber,
   LONG   directoryNumber,
   BYTE   *pathString,
   LONG   *pathCount);
```

Parameters

volumeNumber

(IN) Specifies the volume number of the desired path.

directoryNumber

(IN) Specifies the directory number of the desired path.

pathString

(OUT) Receives the NetWare style path string.

pathCount

(OUT) Receives the path count of the returned path string.

Return Values

0	Success
0x009C	Invalid path---directory number and volume pair cannot be found
0xFFFE	The directory number has become invalid
other NetWare errors	

Remarks

The **FEMapVolumeAndDirectoryToPath** function gets a NetWare style path (pathname and path count) from a volume number and directory number.

FEMapVolumeAndDirectoryToPath relies on the current name space setting of the underlying thread. If that name space does not match the name space of the volume and directory to be mapped, the function returns 0x009C. This error can occur, for example, when the directory number comes from a file system monitoring hook, and the associated name space is something other than DOS.

To avoid the 0x009C error, call **FEMapVolumeAndDirectoryToPath** only if the name space of the underlying thread and the name space of the directory to be mapped can be guaranteed to be identical. Otherwise, call **FEMapVolumeAndDirectoryToPathForNS**, which allows you to specify the name space. You can also call **SetCurrentNameSpace** before and after calling **FEMapVolumeAndDirectoryToPath** to set and restore the current name space of the underlying thread.

0xFFFE (-2) is returned when the directory number has become invalid. This error occurs, for example, when the directory number comes from a FSHOOK_PRE_CLOSE file system monitoring hook, and a separate reporting procedure calls **FEMapVolumeAndDirectoryToPath** after the file has already been deleted.

See Also

FEMapHandleToVolumeAndDirectory, **FEMapPathVolumeDirToVolumeDir**, **FEMapVolumeNumberToName**

FEMapVolumeAndDirectoryToPathForNS

Maps a volume number and directory number to a NetWare style path
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

int FEMapVolumeAndDirectoryToPathForNS (
   int    volumeNumber,
   LONG   directoryNumber,
   LONG   nameSpace,
   BYTE   *pathString,
   LONG   *pathCount);
```

Parameters

volumeNumber

(IN) Specifies the volume number of the desired path.

directoryNumber

(IN) Specifies the directory number of the desired path.

nameSpace

(IN) Indicates the nsame space *directoryNumber* is in.

pathString

(OUT) Receives the NetWare-style path string.

pathCount

(OUT) Receives the path count of the returned path string.

Return Values

This function returns a value of 0 if successful. Otherwise, it returns a NetWare error code.

Remarks

The **FEMapVolumeAndDirectoryToPathForNS** function is useful if you are using FS Hooks.

See Also

FEMapHandleToVolumeAndDirectory, **FEMapPathVolumeDirToVolumeDir**, **FEMapVolumeNumberToName**

FEMapVolumeNumberToName

Maps a volume number to a volume name
Local Servers: nonblocking
Remote Servers: blocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

int FEMapVolumeNumberToName (
   int    volumeNumber,
   BYTE   *volumeName);
```

Parameters

volumeNumber

(IN) Specifies the volume number for which the volume name is desired.

volumeName

(OUT) Receives the name of the volume.

Return Values

This function returns a value of 0 if successful. Otherwise, it returns a NetWare error code.

Remarks

The volume name is returned as a length-preceded ASCII string.

NOTE: This function works remotely only on NetWare 3.12 and 4.x servers.

See Also

FEMapVolumeAndDirectoryToPath, **NWGetVolumeName**

FERegisterNSPathParser

Registers a function to convert a pathname in a name space format to the NetWare format (volume number, path, string, path count)

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x

Platform: NLM

SMP Aware: No

Service: File Engine

Syntax

```
#include <nwfile.h>

int FERegisterNSPathParser (
   T_PathParseFunc   parser);
```

Parameters

parser

(IN) Specifies the address of a function to be called by all other functions that require a NetWare style pathname.

Return Values

This function returns a value of 0 if successful. Otherwise, it returns a NetWare error code.

Remarks

When a path parse function has been registered, and conversion of a pathname to NetWare format is required by a function, the registered name space path parser is called in place of the regular NetWare API path parser.

The registered name space path parser must convert a pathname string into a NetWare pathname. A NetWare pathname consists of a path string count and a string of elements (path). The count is the number of elements that are in the path. Each element can be a length-preceded directory or filename. The NetWare path, however, does not contain the server or volume information.

The following is an example of a NetWare path. The path string count is 3; it contains three elements (dir1, dir2, and dir3).

```
\0x3dir1\0x3dir2\0x8filename
```

The prototype for the path parse function is in nwfile.h and is defined as follows:

```
typedef int (*T_PathParseFunc) (
   const char   *inputPath,
   WORD         *fileServerIDp,
   int          *volumeNumberP,
   LONG         *directoryNumberP,
   BYTE         *outPathStringP,
   LONG         *outPathCountP)
```

inputPath

(IN) Input path string to be parsed.

fileServerID

(OUT) File server ID of the server where the file is located.

volumeNumberP

(OUT) Volume number of the file.

directoryNumberP

(OUT) Directory number of the file.

outPathStringP

(OUT) Path string in NetWare format.

outPathCount

(OUT) Path string count.

The current name space must be set to the appropriate name space before you call **FERegisterNSPathParser**. Call the **SetCurrentNameSpace** function to set the current name space. Once the new path parser is registered, functions such as **open** call the new path parser to translate the *path* parameter into its NetWare counterparts.

See Also

SetCurrentNameSpace

FESetCWDnum

Sets the current working directory (CWD) number (the default directory)
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

LONG FESetCWDnum (
   LONG   CWDnum);
```

Parameters

CWDnum

(IN) Specifies the number of the directory that is to become the default directory for the current thread group.

Return Values

This function returns the old CWD number.

Remarks

The **FESetCWDnum** function sets the directory number that is to be used as the default for parsing pathnames that are not full pathnames.

See Also

FEGetCWDnum, **FEGetCWVnum**, **FESetCWVandCWDnums**, **FESetCWVnum**

FESetCWVandCWDnums

Sets the current working volume (CWV) number and the current working directory (CWD) the default volume and directory

Local Servers: nonblocking

Remote Servers: nonblocking

Classification: 3.x, 4.x

Platform: NLM

SMP Aware: No

Service: File Engine

Syntax

```
#include <nwfile.h>

LONG FESetCWVandCWDnums (
   LONG   CWVnum,
   LONG   CWDnum);
```

Parameters

CWVnum

(IN) Specifies the number of the volume that is to become the default volume for the current thread group.

CWDnum

(IN) Specifies the number of the directory that is to become the default directory for the current thread group.

Return Values

This function returns the old CWD number.

Remarks

The **FESetCWVandCWDnums** function sets the volume and directory numbers that are to be used as the defaults for parsing pathnames that are not full volume paths.

See Also

FEGetCWDnum, **FEGetCWVnum**, **FESetCWDnum**, **FESetCWVnum**

FESetCWVnum

Sets the current working volume (CWV) number (the default volume)
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

LONG FESetCWVnum (
   LONG    CWVnum);
```

Parameters

CWVnum

(IN) Specifies the number of the volume that is to become the default volume for the current thread group.

Return Values

This function returns the old CWV number.

Remarks

The **FESetCWVnum** function sets the volume number that is to be used as the default for parsing pathnames that are not full volume paths.

See Also

FEGetCWDnum, FESetCWDnum, FESetCWVandCWDnums

FESetOriginatingNameSpace

Allows the user to set the originating name space of a directory entry
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

LONG FESetOriginatingNameSpace (
   LONG   volumeNumber,
   LONG   directoryNumber,
   LONG   dirNumNameSpace,
   LONG   newNameSpace);
```

Parameters

volumeNumber

(IN) Specifies the number of the volume on which the directory entry is located.

directoryNumber

(IN) Specifies the directory number of the entry to be changed.

dirNumNameSpace

(IN) Specifies the name space number corresponding with *directoryNumber*.

newNameSpace

(IN) Specifies the name space to be the new originating name space on the directory entry.

Return Values

0	Success
-1	Fail
other NetWare errors	

Remarks

FESetOriginatingNameSpace returns errors on 3.x because there is no OS support.

directoryNumber can be an entry number for any loaded name space.

dirNumNameSpace specifies in which name space the *directoryNumber* is located.

FEsopen

Opens a file for shared access
Local Servers: blocking
Remote Servers: blocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File Engine

Syntax

```
#include <nwfile.h>

int FEsopen (
   char  *name,
   int    access,
   int    share,
   int    permission,
   int    flagBits,
   BYTE   dataStream);
```

Parameters

name

(IN) Specifies the name of the file to be opened.

access

(IN) Specifies the access mode of the file.

share

(IN) Specifies the sharing mode of the file.

permission

(IN) Specifies the file permission (if the file is being created).

flagBits

(IN) Specifies the special flags that allow more file flexibility.

dataStream

(IN) Specifies the flag that indicates the data stream under which the file is to be opened.

Return Values

Returns a file handle upon success. Returns a value of -1, and *errno* and *NetWareErrno* are set to the appropriate error codes if errors occur.

Remarks

FEsopen also works on the DOS partition and is a special version of the **sopen** function. Call the **sopen** function if the primary data stream is requested rather than calling **FEsopen**.

FEsopen does not behave identically to the **sopen** function when only the O_CREAT and O_TRUNC bits are passed. You must also pass DELETE_FILES_ON_CREATE_BIT to the *flagBits* parameter in **FEsopen** which allows the file to be deleted and created again.

The access mode is established as a combination of bits found in the FCNTL.H file and follow:

O_RDONLY	The file can only be read.
O_WRONLY	The file can only be written.
O_RDWR	The file can be read or written.
O_APPEND	Records are written to the end of the file.
O_CREAT	If the file does not exist, it is created.
O_TRUNC	Any data in the file is truncated.
O_BINARY	Data is transmitted unchanged. Text mode is not supported.

The sharing mode is established as a combination of bits found in the NWSHARE.H file and follow:

SH_COMPAT	Sets the compatibility mode.
SH_DENYRW	Prevents read or write access to the file.
SH_DENYWR	Prevents write access to the file.
SH_DENYRD	Prevents read access to the file.
SH_DENYNO	Permits both read and write access to the file.

NOTE: If a new file is created, the share flag is ignored.

If **FEsopen** opens a file for compressed file I/O, the file must be opened

in "exclusive mode" with SH_DENYRW. Otherwise,**FEsopen** fails.

The permission mode is established as a combination of bits found in the SYS\STAT.H file and follow:

S_IWRITE	The file is writeable.
S_IREAD	The file is readable.

A value of 0 can be specified to indicate that the file is readable and writeable.

The flag bits are in the nwfile.h file and follow:

DELETE_FILE_ON_CREATE_BIT	If the file already exists, it is deleted allowing the file to be created again.
NO_RIGHTS_CHECK_ON_OPEN_BIT	The rights to the file are not checked when the file is opened.
NO_RIGHTS_CHECK_ON_CREATE_BIT	The rights to the file are not checked when the file is created.
FILE_WRITE_THROUGH_BIT	When a write is performed, the write function does not return until the data is actually written to disk.
ENABLE_IO_ON_COMPRESSED_ DATA_BIT	Any subsequent I/O on this entry is compressed (NetWare 4.x).
LEAVE_FILE_COMPRESSED_ DATA_BIT	After all I/O has been done, leave this file compressed (NetWare 4.x).

NOTE: If the flag is set to ENABLE_IO_ON_COMPRESSED_DATA_BIT or LEAVE_FILE_COMPRESSED_DATA_BIT (can be ORed), the *share* parameter must be set to SH_DENYRW or **FEsopen** fails.

The *dataStream* parameter is a constant defined in nwfile.h indicating which of the data streams (streams of data stored as separate files on the volume) associated with a file stored on a NetWare 3.x or above server is to be opened. The defined data streams are PrimaryDataStream, MACResourceForkDataStream, and FTAMStructuringDataStream.

See Also

close, **sopen**

File System

File System: Guides

File System: Task Guide

Directory

- Allocating a File System Directory Handle
- Accessing a File System Directory Handle
- Combining a Path and Directory Handle
- Accessing File Information for 3.11 and Above
- Accessing File Information for 2.2 and Above

File

- Deleting Files
- Locating File Beginning and Ending
- Converting File Handles
- Disk Space Management
  - Limiting Directory Space
  - Monitoring File Usage

Trustees

- File System Trustee Management
  - Adding and Deleting File System Trustees
  - Scanning File System Trustees

NLM

- Accessing Files on a Server (NLM)
- Purging and Salvaging Files (NLM)

Additional Links

- File System: Functions
- File System: Structures

Parent Topic:

File System: Guides

File System: Concept Guide

Introduction

File System Introduction

Directory

File System Directory Entries

File System Directory Entry Information

File System Directory Entry Attributes

File System Directory Handles

File System File and Directory Paths

File System Wildcard Characters

File System Search Attributes

Directory Entry Information Access

Accessing File Information for 3.11 and Above: Example

Accessing File Information for 2.2 and Above: Example

File

File Access Operations

File I/O Operations

Disk Space Management

File System Inheritance

File System Effective Rights

Trustees

File System Directory Trustees

File System Trustee Management

File System Directory Trustee Rights

NLM

File Information (NLM)

Attributes (NLM)

File Attributes

Extended File Attributes

File System Tables (NLM)

Directory Entry Table

Volume Table

File System Program: Example (NLM)

Functions - Directory

Directory Entry Functions

Directory Handle Functions

Directory Information Functions

Directory Space Functions

Directory Task Functions

Functions - File

File Handle Conversion Functions

File Information Functions

File System Trustee Functions

File Task Functions

File Usage Functions

Additional Links

File System: Functions

File System: Structures

Parent Topic:

File System: Guides

File System Directory Entries

Volume Directory Entry Tables contain volume files and directories. Consequently, both files and directories are referred to as directory entries. If additional name spaces are loaded on a volume, a file or directory has a directory entry in each name space. However, DOS is the server's primary name space. Therefore, every file or directory is represented by a DOS directory entry.

File System Directory Entry Information

File System Directory Entry Attributes

Parent Topic:

File System: Guides

File System Directory Handles

Directory Handles identify individual directories.

A NetWare® server maintains a Directory Table for each workstation connection. This table is an array of 256 slots, each of which can point to a volume or a volume and directory path. For the DOS client, the server allocates a directory slot for each drive the workstation maps. The workstation can also request that the server enter a directory slot into the table without a drive mapping. This is important for UnixWare clients, which don't support drive mapping.

For each directory the NetWare server enters into the table, the server returns an index to the workstation. This value (from 1 to 256) is referred to as the directory handle. The handle provides a convenient method for referring to the associated directory.

There are several ways to acquire a directory handle for a given directory path. You can use an existing handle as is, you can modify a handle's associated path, or you can allocate a new handle.

Allocating a File System Directory Handle

Accessing a File System Directory Handle

Parent Topic:

File System: Guides

File System File and Directory Paths

From the client point of view, a complete NetWare® file path includes the names of the NetWare server, the volume, any parent directories, and the file itself. For example, in the following file path FS1 is the server, SYS is the volume, DOC and REPORT are directories, and CHAP1.TXT is the filename:

```
FS1/SYS:DOC/REPORT/CHAP1.TXT
```

NetWare accepts forward slashes or back slashes between the components of a file path.

WARNING: All filenames and path parameters must be consistent with the name space used to access the directory entry. For DOS names, all characters should be upper case. Generally, directory

handles and path names are expected to follow DOS conventions unless you are running a different OS and the corresponding name space is loaded for the specified volume.

Combining a Path and Directory Handle

File System Wildcard Characters

File System Search Attributes

Parent Topic:

File System: Guides

Directory Entry Information Access

How you access directory entry information depends on which version of NetWare® is running on the server.

NetWare 3.11 introduced multiple name space support to the NetWare file system. The inherited rights mask replaced the maximum rights mask and was applied to files as well as directories. The set of trustee rights was modified and additional file attributes were added.

The other functions used to access directory entry information for NetWare servers 2.2 and above return a subset of the 3.11 directory entry information. Therefore, additional functions are provided to access the rest of the available information for 3.11 servers and above.

Accessing File Information for 3.11 and Above

Accessing File Information for 3.11 and Above: Example

Accessing File Information for 2.2 and Above

Accessing File Information for 2.2 and Above: Example

Parent Topic:

File System: Guides

Disk Space Management

With NetWare® 3.11 and above, you can control the total amount of space available within a directory and monitor usage for each connection:

Limiting Directory Space

Monitoring File Usage

Parent Topic:

File System: Guides

File System Trustee Management

Duplicate functions exist for adding, deleting, and scanning trustees. One group of functions operates both on directories and files; the other operates only on directories.

Adding and Deleting File System Trustees

Scanning File System Trustees

Parent Topic:

File System: Guides

Attributes (NLM)

The file attributes are contained in a 4-byte field within the file's directory entry stored in the volume's DET. The attributes bytes (bytes 0 to 3) consist of flag bits whose settings can be modified.

File Attributes

Extended File Attributes

Parent Topic:

File System: Guides

File System Tables (NLM)

The server maintains several tables relevant to File System. These tables record information about directories, files, and volumes.

Directory Entry Table

Volume Table

Parent Topic:

File System: Guides

File System: Tasks

Accessing a File System Directory Handle

A pair of functions read and modify the file path associated with a directory handle:

NWGetDirectoryHandlePath

NWSetDirectoryHandlePath

For NetWare® 2.2 and below, special functions are included for saving and restoring a directory handle path:

NWSaveDirectoryHandle

NWRestoreDirectoryHandle

These functions aren't needed in NetWare 3.11 or 4.x environments.

Parent Topic:

File System Directory Handles

Accessing File Information for 2.2 and Above

Under NetWare® 2.2, separate requests access information for files and directories. A pair of functions read and set information for 2.2 files and above:

NWIntScanFileInformation2 reads file information and file attributes.

NWSetFileInformation2 modifies file information.

Another pair of functions access what were considered extended attributes under 2.2, such as the transaction bit:

NWGetExtendedFileAttributes2 reads extended file attributes.

NWSetExtendedFileAttributes2 modifies extended file attributes.

A third pair of functions read and set directory information for 2.2 and above:

NWIntScanDirectoryInformation2 returns directory information.

NWSetDirectoryInformation modifies a directory's creation date/time, owner ID, and maximum rights mask.

Parent Topic:

Directory Entry Information Access

Accessing File Information for 3.11 and Above

File System Services provide access to directory entry information in the DOS name space. (Name Space Services provide access to entry information in other name spaces.) A pair of functions read and set directory entry information:

NWIntScanDirEntryInfo reads directory entry information.

NWSetDirEntryInfo modifies directory entry information.

These functions operate on NetWare® 3.11 and above only. They use three structures to pass directory entry information across the NetWare interface: NWENTRY_INFO, NWFILE_INFO, and NWDIR_INFO.

The following code calls **NWSetDirEntryInfo** to return some information about either a file or a directory. The command line supplies the directory path and search string, and also indicates whether to scan for files or directories. For example, if you want file directory entry information and PROG is the name of the executable, FS1 is the file server, DIR1 is the directory, and *.* is the search string for files, the command line would be:

```
PROG FS1:\DIR1 *.*
```

If you want directory information the command line would be:

```
PROG FS1:\DIR1 * /d
```

NWParseNetWarePath finds the connection handle, and **NWAllocTemporaryDirectoryHandle** gets a directory handle to the input path. **NWSetDirEntryInfo** is then called until it returns an error. Results are displayed for each entry found. The inherited rights mask is shown for directories, and the file attributes are shown for files.

Parent Topic:

Directory Entry Information Access

Accessing Files on a Server (NLM)

Most NetWare® File functions identify files by a file path. The file path can be an absolute with a volume name or it can be relative to the current working directory (CWD):

Absolute Path---Specify the entire path to the target directory or file as the *pathName* parameter.

Relative Path---Specify a current working directory (CWD) using **chdir**. Then specify a directory or file path as the *pathName* parameter. The full path to the target directory or file is the concatenation of the CWD parameter followed by the *pathName* parameter.

File Services functions do not require a server name as a parameter. The target server is always the server to which the NLM™ application is currently logged in (or connected in the case of the local server).

File paths can be up to 255 bytes and must be NULL-terminated. When specifying a file to a File Services function, format the file path as follows:

volume:directory\...\directory\filename

The volume name can be up to 16 characters long and must include a terminating colon (:). The name cannot include spaces or the following characters:

*	Asterisk
?	Question mark
:	Colon
\	Backslash
/	Slash

Filenames and directory names on the network are represented as strings with periods embedded as normal characters. Filenames and directory names can be from 1 to 8 characters and can include a 1 to 3 character extension.

Some NetWare File functions accept wildcard characters in filenames. NetWare supports a larger set of wildcard characters than does DOS.

The following wildcard characters can be used:

*	An asterisk matches zero or more characters. The pattern * therefore matches any string without an extension. The pattern *.* matches anything.

The network wildcard substitution algorithm is implemented as follows:

All characters except the wildcard characters are treated as normal characters.

In a search pattern, the wildcard characters must match the characters

recorded in the file and directory names on the network.

Parent Topic:

File System: Guides

Adding and Deleting File System Trustees

To add to or delete from a file or directory's trustee list, you supply the path specification and a trustee object ID. When adding a trustee you also specify the trustee's rights mask. Only static objects can be added as trustees. If the added object is a trustee already, the trustee's current rights mask is replaced by the new one.

Parent Topic:

File System Trustee Management

Allocating a File System Directory Handle

Directory handles can be permanent or temporary. A temporary handle is deleted as soon as the process that allocated the handle terminates. Permanent handles persist until the connection is closed or a process specifically deallocates them.

Separate functions allocate temporary and permanent directory handles:

NWAllocPermanentDirectoryHandle

NWAllocTemporaryDirectoryHandle

Call **NWDeallocateDirectoryHandle** to deallocate a directory handle. It is especially important to deallocate permanent handles since they can remain after your application terminates.

Parent Topic:

File System Directory Handles

Combining a Path and Directory Handle

Many functions allow you to combine a path with a directory handle to specify a file or directory. If the directory handle parameter is a nonzero value, these functions generally interpret the path relative to the directory associated with the handle. Including a directory handle with a file operation can reduce the amount of space required to store the path variable.

Parent Topic:

File System File and Directory Paths

Converting File Handles

The two basic types of file handles generated in the network environment are local file handles and NetWare® file handles. Local file handles are created and accessed by the local OS running on an individual workstation. NetWare file handles are created for files on the network and are accessed by the NetWare OS. Two functions convert these two types of file handles from one form to the other:

NWConvertFileHandle

NWConvertHandle

NWConvertFileHandle converts a file handle allocated by a local OS to a four-byte or six-byte NetWare file handle. Along with returning the NetWare handle, this function also returns the references of the connection containing the NetWare handle. **NWConvertFileHandle** does not create a NetWare file handle, rather it returns an existing NetWare handle. Therefore the function will fail if the local file handle is not associated with a NetWare file.

NWConvertHandle creates a local file handle from a NetWare file. This function should be called only once per file because it creates a new local file handle and allocates resources each time it is called. The local file handle should be closed using the local OS's close file call.

Parent Topic:

File I/O Operations

Deleting Files

NetWare® files can be deleted on a server using **NWIntEraseFiles**.

Parent Topic:

File Access Operations

Limiting Directory Space

NetWare® 3.11, 3.12 and 4.x servers let you restrict the amount of space allocated to a directory. Directory space limits are specified in 4K blocks. A pair of functions read and set directory space limits:

NWGetDirSpaceLimitList returns the space limit for a directory.

NWSetDirSpaceLimit sets a directory's space limit.

Parent Topic:

Disk Space Management

Locating File Beginning and Ending

The beginning and ending of NetWare® files can be located using **lseek** found with most C compilers.

Parent Topic:

File System: Guides

Monitoring File Usage

File System includes two functions that monitor file usage on a connection basis:

NWScanConnectionsUsingFile scans for a list of connections using a specified file. It returns CONNS_USING_FILE to give the various counts for the file, such as the use count and the open count. For each connection accessing the file, the task number, lock status, and access control are also included.

NWScanOpenFilesByConn2 scans for a list of files opened by a specified connection. It returns an OPEN_FILE_CONN structure identifying the file, and includes information such as the lock status and access control.

These functions are compatible with NetWare® 2.2 and above although there are some differences in the information returned across versions.

Parent Topic:

Disk Space Management

Purging and Salvaging Files (NLM)

An application can mark files for deletion with **remove** or **unlink**. These functions cause files to be marked for deletion. A file marked for deletion is not automatically erased until another file needs the space it occupies. The NetWare® 3.x and 4.x OS saves deleted files (and all information about those files) in their original directory until the server runs out of disk allocation blocks on the volume or until the files marked for deletion are purged.

The **SalvageErasedFile** function can be used to salvage a file that has been

marked for deletion. The **PurgeErasedFile** function can be used to permanently delete a file marked for deletion. Files deleted with **PurgeErasedFile** cannot be recovered.

See Salvaging Files: Example.

Parent Topic:

File System: Guides

Scanning File System Trustees

You can scan for trustees across multiple directories. When you scan for trustees, trustee information is returned as an array of TRUSTEE_INFO. (NWIntScanForTrustees nests this structure within NWET_INFO.) Information for up to 20 trustees can be returned per iteration.

Parent Topic:

File System Trustee Management

File System: Examples

Accessing File Information for 3.11 and Above: Example

NOTE: This example assumes entry names are being returned in the DOS name space.

Accessing File and Directory Information (3.11 and above)

```
#define N_PLAT_DOS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <nwfse.h>
#include <nwdentry.h>
#include <nwdpath.h>
#include <nwmisc.h>
#include <nwalias.h>

void main(int argc, char *argv[ ])
{
   nuint16         attrs;
   NWCONN_HANDLE   conn;
   NWDIR_HANDLE    dirHandle1, dirHandle2;
   NWENTRY_INFO    e;
   nuint32         iterHandle = -1L;
   NWCCODE         ccode     = 0;
   char            dirPath[256], name[14];

   if (argc < 2) {
      printf("\nUsage: INFO <directory path> <search string>
              </d>");
      exit(1);
   }
   /* argv[1] is the directory path, argv[2] is the search string */
   strupr(argv[1]);
   if (argc > 2)
      strupr(argv[2]);
   if (argc > 3)
      attrs = SA_SUBDIR_ONLY;
   else
      attrs = SA_ALL;
```

```
   ccode = NWCallsInit(NULL, NULL);
   if(ccode)
      exit(1);

   ccode = NWParseNetWarePath(argv[1], &conn, &dirHandle1, dirPath);
   if(ccode)
     exit(1);

   ccode = NWAllocTemporaryDirectoryHandle(conn, dirHandle1, dirPath,
                &dirHandle2, NULL);
   iterHandle = -1L;
   for(; ;)
   {
      ccode = NWScanDirEntryInfo(conn, dirHandle2, attrs,
             &iterHandle, (unsigned char *)argv[2], &e);
      if (ccode)
         exit(1);
      strncpy(name, (char *)e.name, 12);
      name[12] = '\0';
      printf("%-14s ", name);
      if (attrs == SA_SUBDIR_ONLY)
      {
         printf("Inherited Rights Mask: [%c%c%c%c%c%c%c%c]\n",
           (e.info.dir.inheritedRightsMask & TR_SUPERVISOR)   ? 'S'
           (e.info.dir.inheritedRightsMask & TR_READ)         ? 'R'
           (e.info.dir.inheritedRightsMask & TR_WRITE)         ? 'W'
           (e.info.dir.inheritedRightsMask & TR_CREATE)         ? 'C'
           (e.info.dir.inheritedRightsMask & TR_DELETE)         ? 'D'
           (e.info.dir.inheritedRightsMask & TR_ACCESS_CTRL)   ? 'A'
           (e.info.dir.inheritedRightsMask & TR_FILE_SCAN)    ? 'F'
           (e.info.dir.inheritedRightsMask & TR_MODIFY)       ? 'M'
      }
      else
      {
         printf("[%s %s %s %s %s %s %s %s %s %s %s %s %s %s]\n",
            (e.attributes & A_READ_ONLY)         ? "Ro"  : "Rw",
            (e.attributes & A_HIDDEN)            ? "H"   : "-",
            (e.attributes & A_SYSTEM)            ? "Sy"  : "--",
            (e.attributes & A_EXECUTE_ONLY)      ? "Xo"  : "--",
            (e.attributes & A_DIRECTORY)         ? "Dir" : "---",
            (e.attributes & A_NEEDS_ARCHIVED)    ? "A"   : "-",
            (e.attributes & A_SHAREABLE)         ? "Sh"  : "--",
            (e.attributes & A_TRANSACTIONAL)     ? "T"   : "-",
            (e.attributes & A_INDEXED)           ? "I"   : "-",
            (e.attributes & A_IMMEDIATE_PURGE)   ? "P"   : "-",
            (e.attributes & A_RENAME_INHIBIT)    ? "RI"  : "--",
            (e.attributes & A_DELETE_INHIBIT)    ? "DI"  : "--",
            (e.attributes & A_COPY_INHIBIT)      ? "CI"  : "--",
            (e.attributes & A_FILE_MIGRATED)     ? "FM"  : "--");
      }
   }
}
```

Parent Topic:

Directory Entry Information Access

Related Topics:

Accessing File Information for 3.11 and Above

Accessing File Information for 2.2 and Above: Example

The following code calls **NWIntScanFileInformation2** to display the file name, size, and creation date for a specified file. Command line parameters supply the file path. **NWParseNetWarePath** parses the path input and finds the connection handle and the current directory handle. **NWAllocTemporaryDirectoryHandle** allocates a new handle to the path input. **NWIntScanDirEntryInfo** could have been used instead of **NWIntScanFileInformation2** to return the same results.

Accessing File Information (2.2 and above)

```
#define N_PLAT_DOS

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <string.h>
#include <time.h>
#include <nwdpath.h>
#include <nwdentry.h>
#include <nwfile.h>
#include <nwmisc.h>

void main(int argc, char *argv[ ])
{
   NW_FILE_INFO2   info;
   NWDIR_HANDLE    dirHandle;
   NWCONN_HANDLE   conn;
   NWCCODE         ccode;
   NW_DATE         sDate;
   NW_TIME         sTime;
   nuint8          iterHandle[9];
   nstr8           newPath[256];

   if (argc != 2) {
      printf("Usage:   LIST <filepath>\n");
      printf("Example: LIST SYS:SYSTEM\\*\n");
      exit(1);
   }
   /* argv[1] is the file path */
   strupr(argv[1]);
```

```
   ccode = NWCallsInit(NULL, NULL);
   if(ccode)
      exit(1);

   ccode = NWParseNetWarePath(argv[1], &conn, &dirHandle, newPath);
   if(ccode)
      exit(1);

   *(LONG *)iterHandle = -1L;
   for(; ;)
   {
      ccode = NWScanFileInformation2(conn, dirHandle, newPath,
              SA_ALL,(unsigned char *)iterHandle, &info);
      if (ccode)
         exit(1);
      printf("%-13s %10ld", info.fileName, info.fileSize);
      NWUnpackDateTime(info.creationDate, &sDate, &sTime);
      printf("\t%2d-%2d-%d\n", sDate.month, sDate.day, sDate.year);
   }
}
```

Parent Topic:

Directory Entry Information Access

Related Topics:

Accessing File Information for 2.2 and Above

File System Program: Example (NLM)

The following example illustrates how **SalvageErasedFile**, **PurgeErasedFile**, and **ScanErasedFiles** can be used.

File System Programming

```
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <fcntl.h>
#include <share.h>
#include <direct.h>
#include <nwfile.h>
#include <nwdir.h>
#include <dos.h>
#include <string.h>
#include <conio.h>
main()
{
   int               rc;
   int               commandChar,purgeAllFlag;
   char              newFileName[100];
```

```
char              fullPathName[200];
struct _DOSTime   filTim;
struct _DOSDate   filDat;
char              scanDirectory[100], *charP;
long              nen;
DIR               dirP;
nen = -1;
purgeAllFlag = 0;
printf("Directory to scan: ");
gets(scanDirectory);
if(!scanDirectory[0])
{
   scanDirectory[0] = '\\';
   scanDirectory[1] = 0;
}

clrscr();
printf("  filename      Size      Attr      Date       Time\
Ser#   Vol#   Seq#\r\n");
printf("------- ----- ----- ---- ----- ---- ---- ----\r\n");
gotoxy( 0, 24);
printf("A - purge all;  P - purge; S - salvage;
   <enter> - next file; X - exit" );
SetScreenRegionAttribute( 24, 1, 0x70 );
gotoxy( 0, 2);
while (!ScanErasedFiles( scanDirectory, &nen, &dirP))
{
   filTim = *(struct _DOSTime *) &dirP.d_time;
   filDat = *(struct _DOSDate *) &dirP.d_date;
   strcpy(newFileName,dirP.d_name);
   if ( (charP = strchr( newFileName, '.' ) ) == NULL)
   charP = " ";
   *charP++ = 0;              /* overwrite period with a zero */

   if (wherey() == 24)
   {
   ScrollScreenRegionUp( 2, 22 );
   gotoxy( 0, 23 );
   }

   printf("%-8.8s %-3.3s   %8d   0x%04x   %02u/%02u/%02u\
   %02u:%02u:%02u   %4d   %4d   %4d\r\n",
   newFileName, charP, dirP.d_size, dirP.d_attr,
   filDat.month, filDat.day, filDat.yearsSince80+80,
   filTim.hour, filTim.minute, filTim.bisecond,
   dirP.d_ino, dirP.d_dev, nen & 0xFFFFFF);
   SetScreenRegionAttribute( wherey()-1, 1, 10 );

   //underline current file
   SetScreenRegionAttribute( wherey()-2, 1, 7 );

   //un-underline prev file
```

```
            if(!purgeAllFlag)
            {
               commandChar = getch();
               if(commandChar == 'A')
               purgeAllFlag = 1;
            }

            if((commandChar == 'p') || (commandChar == 'P') ||
            {
               strcpy(fullPathName,scanDirectory);
               if(scanDirectory[1])
               strcat(fullPathName,"\\");
               strcat(fullPathName,dirP.d_name);
               if (rc = PurgeErasedFile( fullPathName, nen ))
               printf("Could Not Purge File %s;error = %d\r\n",
               fullPathName,rc);
            }
               else if((commandChar == 's') || (commandChar == 'S'))
                  if (wherey() == 24)
                  {
                     ScrollScreenRegionUp( 2, 22 );
                     gotoxy( 0, 23 );
                  }
               printf("new filename: ");
               gets(newFileName);
               strcpy(fullPathName,scanDirectory);
               if(scanDirectory[1])
                  strcat(fullPathName,"\\");
               strcat(fullPathName,dirP.d_name);
               if (rc = SalvageErasedFile( fullPathName, nen, newFileName))
                  printf("Could Not Salvage File %s; error = %d\r\n",
               dirP.d_name,rc);
            }

            else if((commandChar == 'x') || (commandChar == 'X'))
            {
               break;
            }
         }
      }
```

Parent Topic:

File System: Guides

File System: Concepts

Directory Entry Functions

These functions access directory entry information. Some of these functions have older versions that are being phased out. Although both work, Novell® recommends using the newer version.

Function	Comment
NWIntMoveDirEntry	Moves or renames a directory entry (file or directory) on the same server.
NWIntScanDirectoryInformation2	Returns directory information for a directory specified by the connection handle.
NWIntScanDirEntryInfo	Obtains information about 3.x and 4.x directory entries (files or directories.
NWIntScanExtendedInfo	Scans directory for the extended file information.
NWIntScanDirEntryInfo	Scans a 3.11 directory for directory entry information.
NWIntScanExtendedInfo	Scans a directory for extended directory entry information.
NWSetDirEntryInfo	Modifies information for a directory entry.
NWIntMoveDirEntry	Moves or renames a directory entry. The destination must be on the same NetWare® server. (For 2.2 servers this function operates on files only.)

Parent Topic:

File System: Guides

Directory Entry Table

To record information about directories and files, a server maintains a DET. The DET consists of several types of 128-byte entries, including directory nodes, file nodes, and trustee nodes.

A directory node includes the following information about a directory: directory name, attributes, inherited rights mask, creation date and time, creator's object ID, a link to the parent directory, and a link to a trustee node (if one exists). It also includes a name space indicator, last archived date and time, last modification date and time, up to 8 trustee object IDs, up to 8 trustee rights masks.

A file node includes the following information about a file: filename, attributes, file size, creation date and time, deletion date and time, owner's object ID, object ID of the object that performed the last deletion, object IDs of up to 6 trustees, trustee rights mask for up to 6 trustees, inherited rights mask, last-accessed date, last-updated date and time, and a link to a directory.

A trustee node includes the following information: the object IDs of 2 to 16 trustees of a directory linked to the trustee node, 2 to 16 corresponding trustee rights masks, a link to a directory, and a link to the next trustee node (if one exists).

Parent Topic:

File System Tables (NLM)

Directory Handle Functions

These functions read and manipulate directory handles. Note many of the functions work with both regular and short directory handles.

Function	Comment
NWAllocPermanentDirectoryHandle	Allocates a permanent directory handle and returns the caller's effective rights to the associated directory.
NWAllocTemporaryDirectoryHandle	Allocates a temporary directory handle and returns the caller's effective rights to the associated directory.
NWDeallocateDirectoryHandle	Deallocates a directory handle.
NWGetDirectoryHandlePath	Returns the path name of the directory associated with the given directory handle.
NWRestoreDirectoryHandle	Restores a directory handle from its saved state. [NetWare® 2.2 only]
NWSaveDirectoryHandle	Saves the information necessary to restore a directory handle later. [NetWare 2.2 only]

	[NetWare 2.2 only]
NWSetDirectoryHandlePath	Sets the path name of the directory associated with the given directory handle.

Parent Topic:

File System: Guides

Directory Information Functions

These functions access general directory information.

Function	Comment
NWModifyMaximumRightsMask	Modifies a directory's inherited rights mask.
NWIntScanDirectoryInformation2	Returns directory information for the specified directory.
NWSetDirectoryInformation	Changes information about the specified directory.

Parent Topic:

File System: Guides

Directory Space Functions

These functions access directory space limits and return directory space information.

Function	Comment
NWGetDirSpaceLimitList	Returns the actual space limitations for a directory.
NWGetDirSpaceInfo	Returns directory space information.
NWSetDirSpaceLimit	Limits the space available on a specified directory.

Parent Topic:

File System: Guides

Directory Task Functions

These functions create, delete, and rename directories.

Function	Comment
NWCreateDirectory	Creates a NetWare® directory on the specified NetWare server.
NWDeleteDirectory	Deletes a NetWare directory.
NWRenameDirectory	Renames a NetWare directory.

Parent Topic:

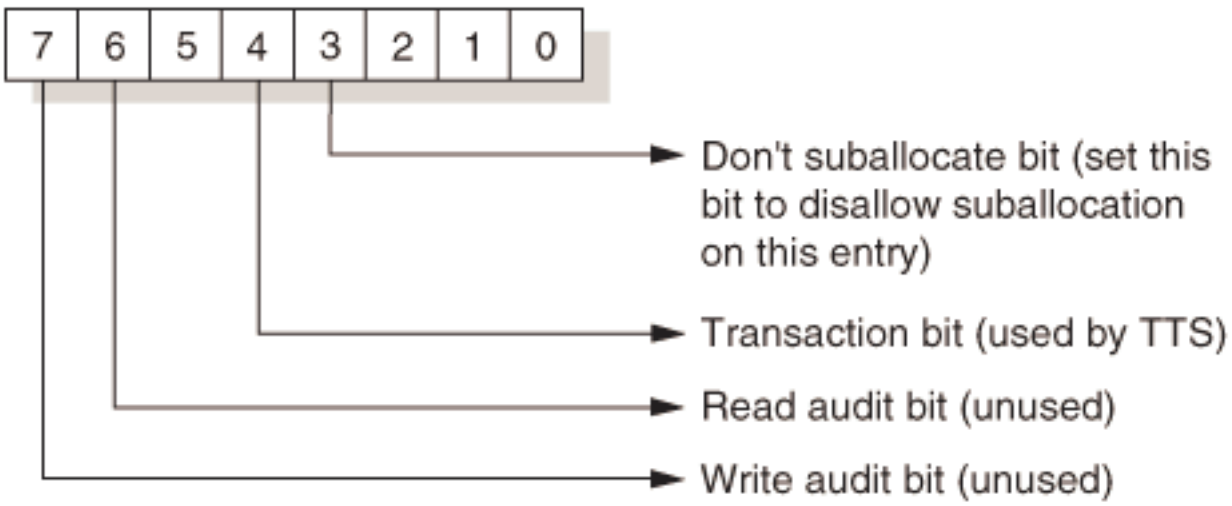
File System: Guides

Extended File Attributes

The **GetExtendedFileAttributes** and **SetExtendedFileAttributes** functions obtain and set the second file attribute byte (byte 1), known in NetWare® 2.x as the extended attributes (and called that here for compatibility), by passing a file path and extended file attributes byte.

The bits in byte 1 have the meanings illustrated in the following figure.

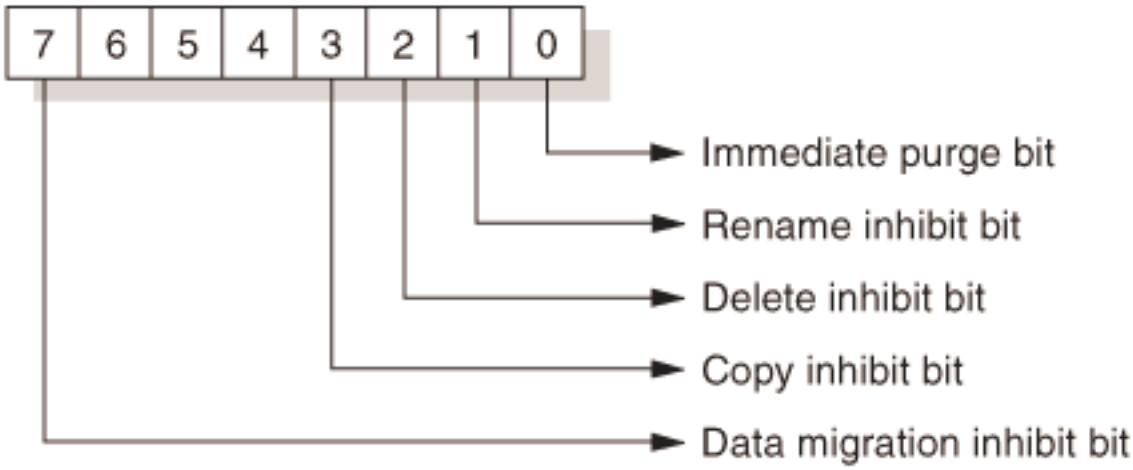
Figure 2. File Attributes Byte 1



The Index file attribute is no longer supported since all the files are automatically indexed when they have 64 or more regular File Allocation Table (FAT) entries and are randomly accessed.

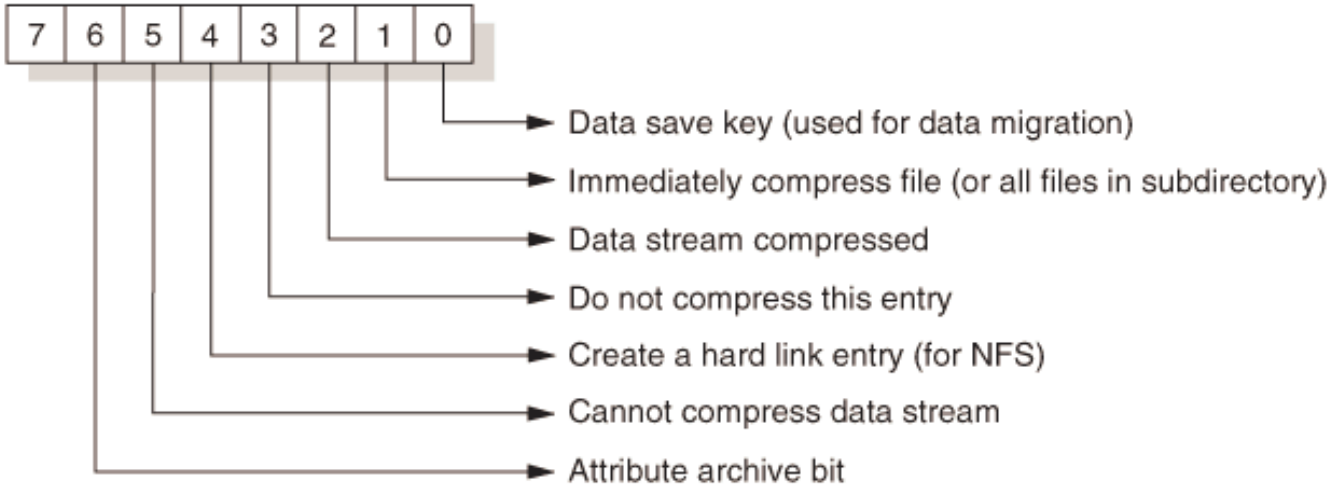
The following figure illustrates the bits defined for byte 2.

Figure 3. File Attributes Byte 2



NetWare 4.x also defines attributes in byte 3 (as shown in the following figure).

Figure 4. File Attributes Byte 3



Parent Topic:

Attributes (NLM)

File Access Operations

NetWare® supports standard DOS services in addition to some specialized functions for accessing NetWare files. Typically, the only difference between accessing a NetWare file and a DOS file is that a NetWare file path includes server and volume names. UnixWare, on the other hand, doesn't support the standard DOS services and must therefore rely more heavily on the specialized functions for file access operations. For high level languages such as C, you can access files using the language's standard I/O functions. Similarly, in assembly language you can use the standard DOS functions.

File System Services supplement standard file IO facilities with functions

that perform single-server operations. These functions can help reduce network traffic since the source and destination of the operations are contained within a single server. For NetWare 3.11 and 4.x these functions operate on a file or a subdirectory:

NWFileServerFileCopy copies a file or a portion of a file to a new location on the same server.

NWRenameFile moves or renames a file on the same server.

NWIntEraseFiles erases NetWare system and hidden files. See Deleting Files.

Parent Topic:

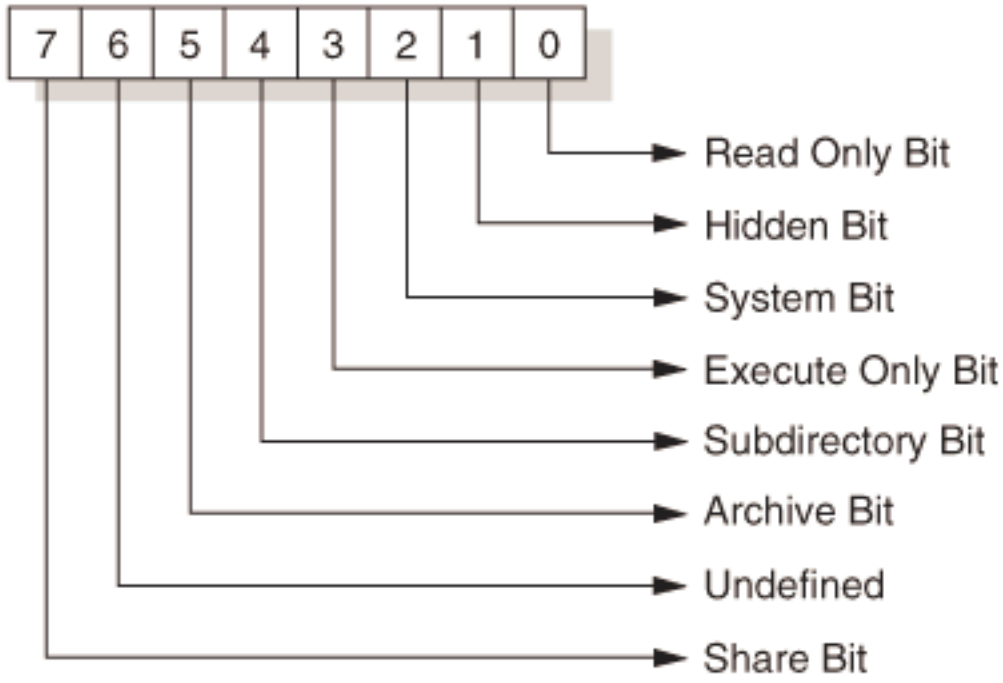
File System: Guides

File Attributes

The low-order file attribute byte contains flag bits similar to the DOS attribute byte. A client must have Modify rights to change the setting of bits in the file attribute bytes.

When set, the bits in the low-order attribute byte (byte 0) have the meanings illustrated in the following figure

Figure 5. File Attributes Byte 0



The following table gives the attribute bits that are set for each possible mode setting (the _A constants are defined in DIRECT.H).

Table auto. Attributes for UNIX Modes

Mode	**Attributes**			
None	_A_EXECUTE	_A_NODELET	_A_NORENAM	_A_SYSTEM
R	_A_RDONLY	_A_NODELET	_A_NORENAM	
W	_A_HIDDEN	_A_NODELET	_A_NORENAM	
X	_A_EXECUTE			
RW	None			
RX	_A_RDONLY	_A_NODELET	_A_NORENAM	
WX	_A_HIDDEN	_A_NODELET	_A_NORENAM	
RWX	None			

The **access** and **chmod** functions indirectly work on the attributes in byte 0. The attribute bits in this byte are used to emulate what is called the mode of the file under UNIX.

File Handle Conversion Functions

These functions provide the ability to convert between local and NetWare® file handles.

Function	Comment
NWConvertFileHandle	Converts a local file handle to a NetWare file handle.
NWConvertHandle	Converts a NetWare file handle to a local file handle.

Parent Topic:

File System: Guides

File Information Functions

These functions search for files, access file information, and monitor file usage. Some of these functions have older versions that are being phased out. Although both work, Novell® recommends using the newer version.

Function	Comment
NWGetSparseFileBitMap	Returns a bit map showing which blocks in a sparse file contain data.
NWIntFileSearchContinue	Performs a search operation for files on the specified volume.
NWIntFileSearchInitialize	Initializes a search operation for files on the specified volume.
NWGetExtendedFileAttributes2	Returns the extended attributes for the specified file.
NWGetFileConnectionID	Returns the connection ID of the NetWare server that owns the specified file handle.
NWIntScanFileInformation2	Scans the specified directory for the specified file (or directory), and returns the associated directory entry information.
NWIntScanFileInformatio	Scans the specified directory for the

NWIntScanFileInformation2	Scans the specified directory for the specified file and returns the file's directory entry information.
NWSetCompressedFileSize	Attempts to set the logical file size for a compressed file.
NWSetExtendedFileAttributes2	Modifies the extended attributes for the specified file.
NWSetFileAttributes	Modifies the attributes for the specified file.
NWSetFileInformation2	Modifies file information for the specified file.

Parent Topic:

File System: Guides

File Information (NLM)

Each network file has directory information associated with it. This data is stored in the server's Directory Entry Table (DET) (see Directory Entry Table).

A file's directory information consists of the file's size, attributes, creation date, date of last access, date and time the file was last modified, and the date and time the file was last archived. It also includes the owner's object ID, object IDs of up to 6 trustees, trustee rights mask for up to 6 trustees, Inherited Rights Mask, and so on. The file attributes contains the information obtained by the NetWare® FLAG utility: read-only versus read/write, sharable versus nonsharable, and so on.

A file's directory information can be set by calling **SetFileInfo**. In addition, **GetExtendedFileAttributes** and **SetExtendedFileAttributes** respectively obtain and set a part of a file's attributes called extended file attributes.

An application can call **SetFileInfo** to set specific file information such as

creationDateAndTime---Creation date of the file (DOS format; 4 bytes)

fileAttributes---File attributes to be assigned to the file

fileOwnerID---Unique Bindery object ID of the file's owner (the name and Bindery object type of the file owner can be obtained by calling **NWGetObjectName**.

lastArchiveDateAndTime---Last archived date and time of the file (DOS format; 4 bytes)

lastUpdateDateAndTime---Last update date and time of the file (DOS format; 4 bytes).

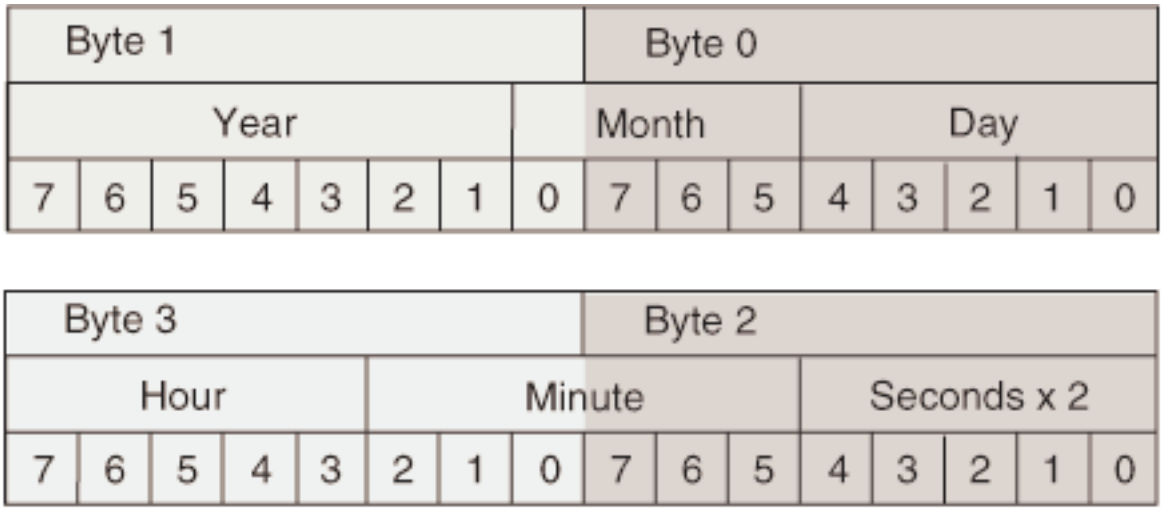
The *creationDateAndTime*, *lastAccessDate*, *lastArchiveDateAndTime*, and *lastUpdateDateAndTime* parameters require a little interpretation. **_ConvertTimeToDOS** and **_ConvertDOSTimeToCalendar** (see Time/Date Manipulation) can be used to manipulate DOS times.

The *creationDateAndTime* parameter consists of 4 bytes indicating the hour, minute, second, year, month, and day that the file was created.

The *lastAccessDate* parameter consists of 2 bytes indicating the year, month, and day that the file was last accessed.

The *lastUpdateDateAndTime* and *lastArchiveDateAndTime* parameters consist of 4 bytes indicating the hour, minute, second, year, month, and day that the file was last modified or archived, respectively. The first 2 bytes of each parameter contain the year, month, and day fields, the same as the *lastAccessDate* parameter. The hour, minute, and second fields are in the second 2 bytes of each parameter. (See the following figure.)

Figure 6. Time Bytes



Parent Topic:

File System: Guides

File I/O Operations

File I/O functions provide the ability to perform the following tasks:

Convert local file handles to NetWare® file handles

Convert NetWare file handles to local file handles

Convert NetWare file handles to local file handles

See Converting File Handles for information on how to perform these tasks.

Parent Topic:

File System: Guides

File System Directory Entry Attributes

Directory entry attributes are commonly known as file flags (though they also can pertain to directories). They have wide influence over the events that can or will be performed on a directory or file entry. The following table lists the attributes and explains their function.

Table auto. Directory Entry Attributes Defined

Attribute	Bit Value	Application	Comment
A_READ_ONLY	0x00000001L	Files only.	Entry can't be written, deleted or renamed.
A_HIDDEN	0x00000002L	Files and directories.	Entry doesn't appear in a normal directory listing.
A_SYSTEM	0x00000004L	Files and directories.	Entry is used by the system and is hidden.
A_EXECUTE_ONLY	0x00000008L	Files only.	Entry can be loaded for execution only once.
A_DIRECTORY	0x00000010L	Files and directories.	Entry is a directory, not a file.
A_NEEDS_ARCHIVED	0x00000020L	Files only.	Entry has been changed since last archived.
A_SHAREABLE	0x00000080L	Files only.	Entry can be opened by multiple clients.
A_DONT_SUBALLOCATE	0x00000800L	Files only.	A file is stored in its own separately allocated memory for ease of access.

A_TRANSACTIONAL	0x00001000L	Files only.	A transaction on the entry is being tracked.
A_INDEXED	0x00002000L	Files and directories.	Not in use. Provided for compatibility only.
A_READ_AUDIT	0x00004000L	Files and directories.	Not in use.
A_WRITE_AUDIT	0x00008000L	Files and directories.	Not in use.
A_IMMEDIATE_PURGE	0x00010000L	Files and directories.	Entry will be purged when deleted.
A_RENAME_INHIBIT	0x00020000L	Files only.	Entry can't be renamed.
A_DELETE_INHIBIT	0x00040000L	Files and directories.	Entry can't be deleted.
A_COPY_INHIBIT	0x00080000L	Files only.	Entry can't be copied.
A_FILE_MIGRATED	0x00400000L	Files only.	Entry has been migrated.
A_DONT_MIGRATE	0x00800000L	Files only.	Entry should not be migrated.
A_IMMEDIATE_COMPRESS	0x02000000L	Files only.	Entry should be compressed when written.
A_FILE_COMPRESSED	0x04000000L	Files only.	Entry is compressed.
A_DONT_COMPRESS	0x08000000L	Files only.	Entry should not be compressed.
A_CANT_COMPRESS	0x20000000L	Files only.	Entry can't be compressed.

Parent Topic:

File System Directory Entries

File System Directory Entry Information

The term "directory entry information" is used loosely to refer to the DOS information associated with a file or directory. The file system uses directory entry information to maintain the file or directory entry. Some of the more significant items included in directory entry information are the following:

Short name

Directory entry attributes

Owner ID

Inherited rights mask

Entry event dates and times

Additional information is also included depending on whether the entry is a file or directory. For example, file size is returned for files and maximum space is returned for directories.

Parent Topic:

File System Directory Entries

File System Directory Trustees

Directory trustees are network users assigned access rights to a directory or file. Trustees are identified by their object ID. Access rights at both the directory and files level are expressed as a bit mask. (NetWare® 2.2 servers don't support trustee assignments at the file level.)

Parent Topic:

File System: Guides

File System Directory Trustee Rights

The following trustee rights are defined for NetWare® 3.11 and above.

0x0001 TR_READ
0x0002 TR_WRITE
0x0004 undefined
0x0008 TR_CREATE
0x0010 TR_DELETE
0x0020 TR_ACCESS_CTRL
0x0040 TR_FILE_SCAN
0x0080 TR_MODIFY
0x0100 TR_SUPERVISOR

With the exception of the TR_SUPERVISOR bit, these rights also apply to

NetWare 2.2. The following table compares the privileges associated with trustee rights when assigned at the directory level and at the file level.

Table auto. Trustee Rights

Right	Directory Level	File Level
TR_READ	Trustee can open and read the directory.	Trustee can open and read the files.
TR_WRITE	Trustee can open and write to the directory.	Trustee can open and write to the file.
TR_CREATE	Trustee can create entries in the directory.	Trustee can salvage the file after deletion.
TR_ERASE	Trustee can remove entries from the directory.	Trustee can erase the file.
TR_ACCESS_CTRL	Trustee can grant trustee rights and modify inheritance for the directory.	Trustee can grant trustee rights and modify inheritance for the file.
TR_FILE_SCAN	Trustee can scan for directory entries.	Trustee can see the file when scanning.
TR_MODIFY	Trustee can modify directory attributes and rename entries.	Trustee can modify the file's attributes (but not its content).
TR_SUPERVISOR	Trustee has all rights to the directory.	Trustee has all rights to the file.

Parent Topic:

File System: Guides

File System Effective Rights

Effective rights take into account a trustee's assigned rights, inherited rights, and security equivalences to find the rights a trustee can exercise. To find the effective rights for a file or directory under your current object ID, call **NWGetEffectiveRights**.

An assigned rights mask takes precedence over any inherited rights. It can remove rights that would have been inherited or grant new rights that would not have been inherited. A trustee's assigned rights are not affected by an Inherited Rights Mask. Consequently, the computation of effective rights depends on whether rights are assigned or inherited:

If a trustee has an assigned rights mask, effective rights are computed by ORing the trustee's rights mask with any assigned rights mask of objects that the trustee is equivalent to in the bindery.

If the trustee does not have assigned rights (either directly or through equivalence) in a given directory, the trustee inherits rights assigned (directly or through equivalence) in a superior directory. These rights are limited by the Inherited Rights Mask. The effective inherited rights are computed by ORing the trustee's inherited rights with any equivalent inherited rights, then ANDing the result with the Inherited Rights Mask.

Parent Topic:

File System: Guides

File System Inheritance

Rights assigned to a trustee in the parent directory apply to all subordinate directories. This is referred to as inheritance. The trustee does not need to appear in the trustee list of a subordinate directory to receive these rights.

There are two ways to block inheritance:

The trustee may be assigned new rights in a subordinate directory (thus overriding the inherited rights).

The Inherited Rights Mask for the directory (or file) can be modified to exclude specific rights.

When a file or directory is created, its Inherited Rights Mask includes all rights. Any rights removed from the inherited rights mask can't be inherited. An exception is the TR_SUPERVISOR bit, which can't be masked by an Inherited Rights Mask.

The Inherited Rights Mask is stored with directory entry information. See Directory Entry Information Access for a description of functions that read and modify this information.

Parent Topic:

File System: Guides

File System Introduction

File System functions enable developers to manipulate NetWare® file system information. The principle operations performed by these functions include:

Accessing files

Accessing directory entry information

Managing disk space

Monitoring file usage

Managing trustees

Although this chapter generally notes the differences between overlapping functions, it's important for developers to be aware of compatibility issues affecting specific functions. To verify a function's compatibility, see the specific reference for that function.

Functions beginning with NWInt, such as **NWIntScanFileInformation2**, support wildcard augmentation of filename parameters. Functions ending with integers such as 2 or 3 include support for more recent file system features (such as long names).

For a description of structures and other data definitions that relate to this chapter, see File System: Structures.

Parent Topic:

File System: Guides

File System Search Attributes

Functions operating on directory entries typically include a search attribute. The attribute specifies the type of entries to include in the operation. The search attribute lets you include system and hidden files and files in subdirectories.

For functions that can operate on both directories and files, typically do one to the exclusion of the other. For these functions, the search attribute lets you specify whether to operate on files or directories. Below are the possible bits defined by the search attribute:

0x0000 SA_NORMA
0x0002 SA_HIDDEN
0x0004 SA_SYSTEM
0x0010 SA_SUBDIR_ONLY
0x8000 SA_SUBDIR_FILES
0x8006 SA_ALL

Parent Topic:

File System File and Directory Paths

File System Trustee Functions

These functions operate on directories or files and so are oriented more toward NetWare® 3.11 and above. On NetWare 2.2 servers, these functions operate on directories only; however, they don't verify whether the input

path specifies a file or directory.

NWAddTrustee

NWDeleteTrustee

NWIntScanForTrustees

These functions operate on directories only, and so are oriented toward NetWare 2.2. These functions aren't able to read or set the TR_SUPERVISOR bit.

NWAddTrusteeToDirectory

NWDeleteTrusteeFromDirectory

NWScanDirectoryForTrustees2

NWIntScanForTrustees

Parent Topic:

File System: Guides

File System Wildcard Characters

Many functions accept wildcard characters within a filename parameter. For example, with **NWIntEraseFiles** the file path can include wildcard characters, in which case a single request is able to erase multiple files. The following table shows the wildcard characters supported by NetWare®.

Table auto. NetWare Wildcard Characters

Character	Wildcard	Match
*	Asterisk	Zero or more characters.
?	Question mark	Any single character.

Parent Topic:

File System File and Directory Paths

File Task Functions

These functions erase, copy, and rename files on a NetWare® server. Some of these functions have older versions that are being phased out. Although both work, Novell® recommends using the newer version.

Function	Comment
NWIntEraseFiles	Deletes NetWare files from a server.
NWFileServerFileCopy	Copies from one file to another. The source and target directories must be on the same NetWare server.
NWIntEraseFiles	Deletes NetWare files from the server.
NWIntFileSearchContinue	Iteratively retrieves all directory entries matching *searchPath*.
NWRenameFile	Moves or renames a file.

Parent Topic:

File System: Guides

File Usage Functions

These functions return file usage statistics.

Function	Comment
NWScanConnectionsUsingFile	Returns a list of workstation connection numbers for connections using the specified file.
NWScanOpenFilesByConn2	Returns information for files currently opened by the specified connection.

Parent Topic:

File System: Guides

Volume Table

To record information about volumes, a server maintains a Volume Table that includes the number of volumes mounted in the server, the name, size, and other information pertaining to each volume. Functions that return information about volumes access the Volume Table.

Parent Topic:

File System Tables (NLM)

File System: Functions

access

Determines whether a file or directory exists and if it can be accessed
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <unistd.h>

int access (
   const char  *path,
   int          mode);
```

Parameters

path

(IN) Specifies the string containing the path that includes the file or directory to be accessed (maximum 255 characters, including the NULL terminator).

mode

(IN) Specifies the access permission mode for the file.

Return Values

Returns 0 if the file or directory exists and can be accessed with the specified mode. Otherwise, it returns a value of -1. If an error occurs, the errno parameter is set.

Remarks

access also works on the DOS partition.

access determines if the file or directory specified by the *path* parameter exists and if it can be accessed with the file permission given by the*mode* parameter.

When the *mode* parameter is 0, only the existence of the file is verified. The read and/or write and/or execute permission for the file can be determined when the bits of the *mode* parameter are a combination of the following:

0	F_OK: File existence
1	X_OK: Execute permission
2	W_OK: Write permission
4	R_OK: Read permission

The result is dependent on the current connection number.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to this function.

NOTE: For NetWare® versions before 4.x, **access** only works with DOS name space for remote servers.

See Using access(): Example.

See Also

chmod, **fstat**

chdir

Changes the current working directory to the specified path name
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <unistd.h>

int chdir (
   const char  *pathname);
```

Parameters

pathname

(IN) Specifies the buffer containing the directory path (can include a volume name).

Return Values

Returns a value of 0 if successful, nonzero otherwise. If an error occurs, errno and NetWareErrno are set.

Remarks

chdir causes all threads in the current thread group to have a new current working directory. The *pathname* parameter can be either relative to the current working directory or it can be an absolute path name.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to **chdir**.

NOTE: For NetWare versions before 4.x, **chdir** only works with DOS name space for remote servers.

See Also

getcwd, **mkdir**, **rmdir**

chmod

Changes the file access mode
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <stat.h>

int chmod (
   const char  *path,
   int          mode);
```

Parameters

path

(IN) Specifies the string containing the path that includes the file whose access mode is to be modified (maximum 255 characters, including the NULL terminator).

mode

(IN) Specifies the access permission mode for the file.

Return Values

Returns a value of 0 if successful, -1 otherwise. If an error occurs, errno is set.

Remarks

chmod also works on the DOS partition.

The current connection must have modify permission to the specified file to call **chmod**.

The various mode settings are given in the SYS\STAT.H header file. The access permissions for the file are specified as a combination of bits defined in the SYS\STAT.H header file.

S_IWRITE	The file is writeable

S_IREAD	The file is readable

Alternatively, zero can be specified to indicate that the file is readable and writeable.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input.

NOTE: For NetWare versions before 4.x, **chmod** only works with DOS name space for remote servers.

See Also

fstat, stat (Function)

closedir

Closes a specified directory
Local Servers: nonblocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <dirent.h>

int closedir (
   DIR  *dirP);
```

Parameters

dirP

Specifies the directory to be closed.

Return Values

0	0x00	ESUCCESS
22	0x16	EBADHNDL
NetWare Error		UNSUCCESSFUL

Remarks

closedir closes the directory specified by the *dirP* parameter and frees the memory allocated by the **opendir** function. All open directories are automatically closed when an NLM™ application is terminated.

See Also

opendir, **readdir**

FileServerFileCopy

Copies a file, or a portion of a file, to another file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwfile.h>

int FileServerFileCopy (
   int    sourceFileHandle,
   int    destinationFileHandle,
   LONG   sourceFileOffset,
   LONG   destinationFileOffset,
   LONG   numberOfBytesToCopy,
   LONG  *numberOfBytesCopied);
```

Parameters

sourceFileHandle

(IN) Specifies the file handle of the source file.

destinationFileHandle

(IN) Specifies the file handle of the destination file.

sourceFileOffset

(IN) Specifies the offset (in bytes) in the source file where copy should begin.

destinationFileOffset

(IN) Specifies the offset (in bytes) in the destination file where the data should be copied.

numberOfBytesToCopy

(IN) Specifies the number of bytes to be copied.

numberOfBytesCopied

(OUT) Points to the number of bytes actually copied.

Return Value

0	0x0	ESUCCESS

	0	
1	0x01	ERR_INSUFFICIENT_SPACE
22	0x16	EBADHNDL
131	0x83	ERR_NETWORK_DISK_IO
136	0x88	ERR_INVALID_FILE_HANDLE
147	0x93	ERR_NO_READ_PRIVILEGE
148	0x94	ERR_NO_WRITE_PRIVILEGE_OR_READONLY
149	0x95	ERR_FILE_DETACHED
162	0xA3	ERR_IO_LOCKED

Remarks

An application must pass file handles in the *sourceFileHandle* and *destinationFileHandle* parameters. A file handle can be obtained by calling the **open**, **sopen**, **creat**, or **fileno** function.

To copy from the beginning of the source file to a new file, set the *sourceFileOffset* and *destinationFileOffset* parameters to 0x00.

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

The *numberOfBytesCopied* parameter returns the number of bytes copied between files as a result of calling this function.

On remote servers running NetWare 2.x, **FileServerFileCopy** returns ERR_NO_READ_PRIVILEGE.

See Also

creat, **fileno**, **open**, **sopen**

getcwd

Returns the current working directory of the current thread group
Local Servers: either blocking or nonblocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <unistd.h>

char *getcwd (
   char    *buffer,
   size_t   size);
```

Parameters

buffer

(OUT) Specifies the buffer in which to place the current working directory.

size

(IN) Specifies the length of buffer (including space for the delimiting \0 character).

Return Values

Returns the address of the string containing the name of the current working directory if successful. Otherwise, NULL is returned and errno is set.

Remarks

When the *buffer* parameter is NULL, a string is allocated to contain the current working directory. The string can be freed by calling the **free** function.

Blocking Information Locally, **getcwd** blocks when the *buffer* parameter is NULL and does not block when the *buffer* parameter is not NULL.

See Also

chdir, **free**, **mkdir**, **rmdir**

GetExtendedFileAttributes

Returns the extended attributes for a file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwfile.h>

int GetExtendedFileAttributes (
   char  *filePath,
   BYTE  *extendedFileAttributes);
```

Parameters

filePath

(IN) Points to a string containing the absolute path or path relative to the current working directory of the file for which to get extended file attributes (maximum 255 characters, including the NULL terminator).

extendedFileAttributes

(OUT) Points to the extended attributes.

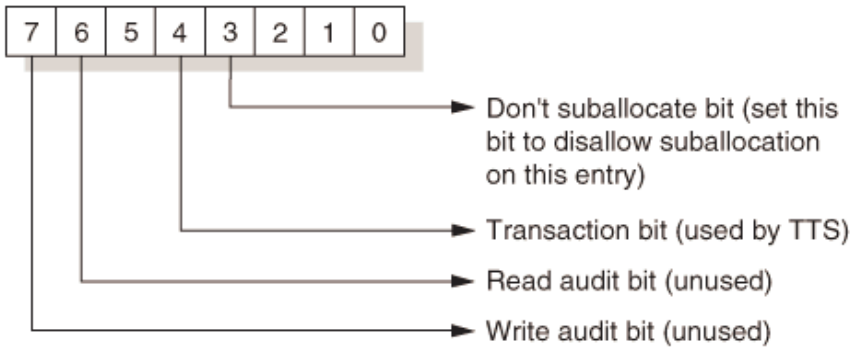
Return Value

0	0x00	ESUCCESS
137	0x89	ERR_NO_SEARCH_PRIVILEGE
156	0x9C	ERR_INVALID_PATH
158	0x9F	ERR_BAD_FILE_NAME
191	0xBF	ERR_INVALID_NAME_SPACE
253	0xFD	ERR_BAD_STATION_NUMBER
254	0xFE	ERR_SPOOL_DIRECTORY_ERROR

4 E

Remarks

GetExtendedFileAttributes returns the value of the first byte of the file attributes, known as the extended attributes byte. The following bits are defined:



NOTE: Do not confuse the file attributes byte with true extended attributes, which can be manipulated with the Extended Attribute functions.

If the transaction bit is set in the *extendedFileAttributes* parameter, NetWare TTS™ software tracks all writes to the file during a transaction. A transaction file cannot be deleted or renamed until the transaction bit is turned off with the **SetExtendedFileAttributes** function.

An application can specify a file in several ways. For example, suppose the full path of the file TARGET.DAT is:

```
SYS:ACCOUNT\DOMEST\TARGET.DAT
```

and the current working directory is SYS:ACCOUNT. The application can specify the partial path, DOMEST\TARGET.DAT, or the full path in the *filePath* parameter.

GetExtendedFileAttributes requires that the current connection have See File rights to the directory where the file resides.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to **GetExtendedFileAttributes**.

NOTE: For NetWare versions before 4.x, **GetExtendedFileAttributes** only works with DOS name space for remote servers.

See Also

SetExtendedFileAttributes

_makepath

Constructs a full NetWare path name
Local Servers: blocking
Remote Servers: N/A
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwdir.h>

void _makepath (
   char        *path,
   const char  *volume,
   const char  *dir,
   const char  *fname,
   const char  *ext);
```

Parameters

path

(OUT) Points to the string containing the full path name.

volume

(IN) Specifies the volume name.

dir

(IN) Specifies the directory name.

fname

(IN) Specifies the base name of the file without an extension.

ext

(IN) Specifies the file name extension.

Return Values

None

Remarks

The NetWare path name is constructed from the components consisting of a volume name, directory path, file name, and file name extension. The full path name is placed in the buffer pointed to by the *path* parameter.

The maximum size required for each buffer is specified by the manifest

constants which are defined in the NWDIR.H file.

```
255   _MAX_PATH
 16   _MAX_VOLUME (volume name length)
255   _MAX_DIR
  9   _MAX_FNAME
  5   _MAX_EXT
```

See Using _makepath and _splitpath: Example.

See Also

_splitpath

mkdir

Creates a new directory
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <stat.h>

int mkdir (
   const char  *pathname);
```

Parameters

pathname

(IN) Specifies the path containing the new directory (either relative to the current working directory or an absolute path name).

Return Values

Returns a value of 0 if successful, nonzero otherwise.

Remarks

mkdir also works on the DOS partition.

The current connection must have Create rights in the parent directory. The inherited rights mask for the new directory is ALL rights.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to **mkdir**.

NOTE: For NetWare versions before 4.x, **mkdir** only works with DOS name space for remote servers.

See Also

chdir, **getcwd**, **rmdir**

NWAddTrustee

Adds a trustee to the list of trustees in a file or directory
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWAddTrustee (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint32         objID,
   nuint16         rightsMask);
```

Pascal Syntax

```
#include <nwdentry.inc>

Function NWAddTrustee
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   objID : nuint32;
   rightsMask : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired directory path (0 if the *path* parameter contains the complete path, including the volume name).

path

(IN) Points to the absolute path (or a path relative to the *dirHandle* parameter) of the directory to which a trustee is being added.

objID

(IN) Specifies the bindery object ID for the object being added as a trustee.

rightsMask

(IN) Specifies the access rights mask being granted to the new trustee.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	NO_MODIFY_PRIVILEGES
0x8990	NO_FILES_AFFECTED_READ_ONLY
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FC	NO_SUCH_OBJECT
0x89FD	BAD_STATION_NUMBER
0x89FF	HARDWARE_FAILURE

Remarks

To modify a trustee rights list, the requesting workstation must have parental rights to the directory or to a parent of the directory.

If the object is already a trustee for the specified directory, the current access mask of the trustee is replaced by the value contained in the *rightsMask* parameter. Otherwise, the object is added as a trustee to the directory with rights equal to the *rightsMask* parameter.

NOTE: 2.x servers do not support file level trustee assignments. **NWAddTrustee** does not check whether the specified path is a file, or a directory path. The input parameters are passed as is to the server, and the server returns an error for a file level input.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 22 13 Add Trustee To Directory
0x2222 22 39 Add Extended Trustee To Directory Or File
0x2222 87 10 Add Trustee Set To File Or Subdirectory

See Also

NWAddTrusteeToDirectory

NWAddTrusteeToDirectory

Adds a trustee to the trustee list in a directory
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWAddTrusteeToDirectory (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint32         trusteeID,
   nuint8          rightsMask);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWAddTrusteeToDirectory
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   trusteeID : nuint32;
   rightsMask : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired directory path (0 if the *path* parameter contains the complete path, including the volume name).

path

(IN) Points to the absolute path (or a path relative to the directory handle) of the directory to which a trustee is being added.

trusteeID

(IN) Specifies the bindery object ID for the object being added as a trustee.

rightsMask

(IN) Specifies the access rights mask the new trustee is being granted.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	NO_MODIFY_PRIVILEGES
0x8990	NO_FILES_AFFECTED_READ_ONLY
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FC	NO_SUCH_OBJECT
0x89FD	BAD_STATION_NUMBER
0x89FF	HARDWARE_FAILURE

Remarks

If the object is already a trustee for the specified directory, the current access mask of the trustee is replaced by the value contained in the *trusteeID*

trusteeID parameter. Otherwise, the object is added as a trustee to the directory and given a rights mask equal to the *trusteeID* parameter.

To modify a trustee rights list, the requesting workstation must have parental rights to the directory or to a parent of the directory.

The object must be static. If the object is dynamic, **NWAddTrusteeToDirectory** will return an error.

NCP Calls

0x2222 22 13 Add Trustee To Directory
0x2222 22 39 Trustee Add Ext
0x2222 23 17 Get File Server Information
0x2222 87 10 Add Trustee Set To File Or Subdirectory

See Also

NWAddTrustee

NWAllocPermanentDirectoryHandle

Allocates a permanent directory handle for a network directory
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWAllocPermanentDirectoryHandle (
   NWCONN_HANDLE         conn,
   NWDIR_HANDLE          dirHandle,
   pnstr8                dirPath,
   NWDIR_HANDLE N_FAR   *newDirHandle,
   pnuint8               effectiveRights);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWAllocPermanentDirectoryHandle
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   dirPath : pnstr8;
   Var newDirHandle : NWDIR_HANDLE;
   effectiveRights : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired directory path.

dirPath

(IN) Points to an absolute directory path (or a path relative to the *dirHandle* parameter) specifying the directory with which the new directory handle is to be associated (optional).

newDirHandle

(OUT) Points to the new directory handle.

effectiveRights

(OUT) Points to the effective rights of the directory trustee connected through the *dirHandle* parameter (optional).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899D	NO_MORE_DIRECTORY_HANDLES
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	INVALID_DRIVE_NUMBER, HARDWARE_FAILURE

Remarks

To deallocate a permanent directory handle, call the **NWDeallocateDirectoryHandle** function.

If more than 255 handles are allocated, **NWAllocPermanentDirectoryHandle** may return a successful code; however, the *dirHandle* parameter will be zero.

NCP Calls

0x2222 22 03 Get Effective Directory Rights
0x2222 22 18 Alloc Permanent Directory Handle
0x2222 23 17 Get File Server Information
0x2222 87 12 Allocate Short Directory Handle

See Also

NWAllocTempNSDirHandle2, **NWAllocTemporaryDirectoryHandle**, **NWDeallocateDirectoryHandle**

NWAllocTemporaryDirectoryHandle

Assigns a temporary directory handle for the current name space
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWAllocTemporaryDirectoryHandle (
   NWCONN_HANDLE         conn,
   NWDIR_HANDLE          dirHandle,
   pnstr8                dirPath,
   NWDIR_HANDLE N_FAR   *newDirHandle,
   pnuint8               rightsMask);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWAllocTemporaryDirectoryHandle
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   dirPath : pnstr8;
   Var newDirHandle : NWDIR_HANDLE;
   rightsMask : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired directory path (or 0 if the *dirPath* parameter points to the complete path, including the volume name).

dirPath

(IN) Points to an absolute directory path (or a path relative to the NetWare directory handle) specifying the directory with which the new directory handle is associated.

newDirHandle

(OUT) Points to the new directory handle.

rightsMask

(OUT) Points to the effective rights of the directory trustee connected through the *newDirHandle* parameter (optional).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899D	NO_MORE_DIRECTORY_HANDLES
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	INVALID_DRIVE_NUMBER, HARDWARE_FAILURE

Remarks

The directory handles allocated by **NWAllocTemporaryDirectoryHandle** are automatically deallocated when the task terminates, or when the **NWDeallocateDirectoryHandle** function is called.

If more than 255 handles are allocated, **NWAllocTemporaryDirectoryHandle**

NWAllocTemporaryDirectoryHandle may return a successful code; however, the *dirHandle* parameter will be zero.

Under DOS and Windows 3.1, the current name space is NW_NS_DOS.

Under OS/2, Windows NT, and Windows95, the current name space is NW_NS_OS2.

NCP Calls

0x2222 22 03 Get Effective Directory Rights
0x2222 22 19 Allocate Temporary Directory Handle
0x2222 23 17 Get File Server Information
0x2222 87 12 Allocate Short Directory Handle

See Also

NWAllocPermanentDirectoryHandle, **NWAllocTempNSDirHandle2**, **NWDeallocateDirectoryHandle**

NWConvertFileHandle

Converts a file handle to a 4- or 6-byte NetWare handle
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWConvertFileHandle (
   NWFILE_HANDLE          fileHandle,
   nuint16                handleType,
   pnuint8                NWHandle,
   NWCONN_HANDLE N_FAR  *conn);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWConvertFileHandle
  (fileHandle : NWFILE_HANDLE;
   handleType : nuint16;
   NWHandle : pnuint8;
   Var conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

fileHandle

(IN) Specifies the name of the local file handle to be converted to a NetWare handle.

handleType

(IN) Specifies the type of handle to create:

4 = Create a 4-byte NetWare handle
6 = Create a 6-byte NetWare handle

NWHandle

(OUT) Points to a 4- or 6-byte NetWare Handle to which the local file handle is being converted.

conn

(OUT) Points to the connection for which the NetWare handle is valid (optional).

Return Values

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

0x0000	SUCCESSFUL
0x0006	INVALID_HANDLE
0x8801	INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE

Remarks

The handle returned by **NWConvertFileHandle** should not be used to call the **NWConvertHandle** function. Otherwise, a new OS file handle will be created.

If **NWConvertFileHandle** is called with only theNETX shell running, INVALID_CONNECTION will be returned. However, the NetWare handle will still be valid and the *conn* parameter will be set to zero.

If NULL is passed in the *conn* parameter, no error will be returned. If a pointer is passed in the *conn* parameter and the shell is running, a valid NetWare handle will be returned as well as 0x8801.

When a connection handle is obtained, a new licensed connection handle will be created. Close the new connection handle by calling the **NWCCCloseConn** function.

NCP Calls

None

See Also

NWConvertHandle

NWConvertHandle

Converts a NetWare handle to a local file handle
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWConvertHandle (
   NWCONN_HANDLE          conn,
   nuint8                 accessMode,
   nptr                   NWHandle,
   nuint16                handleSize,
   nuint32                fileSize,
   NWFILE_HANDLE N_FAR   *fileHandle);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWConvertHandle
  (conn : NWCONN_HANDLE;
   accessMode : nuint8;
   NWHandle : nptr;
   handleSize : nuint16;
   fileSize : nuint32;
   Var fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the connection where the NetWare handle exists to which the local file handle is being converted.

accessMode

(IN) Specifies the type of access the user will have to the newly created file handle.

NWHandle

(IN) Points to the 4- or 6-byte NetWare handle being converted to a local file handle.

handleSize

(IN) Specifies the number of bytes in the NetWare handle; either 4 or 6.

fileSize

(IN) Specifies the number of bytes in the file being converted.

fileHandle

(OUT) Points to the local file handle created by **NWConvertHandle**.

Return Values

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

0x0000	SUCCESSFUL

Remarks

The handle returned by the **NWConvertFileHandle** function should not be used to call **NWConvertHandle**. Otherwise, a new OS file handle will be created.

The file handle returned is appropriate for the platform for which the function is written. The file handle may be used for access to the attribute value including closing the file as well as reading and writing to the file.

The *accessMode* parameter can have the following values:

0x0001	$0001	AR_READ (AR_READ_ONLY)
0x0002	$0002	AR_WRITE (AR_WRITE_ONLY)
0x0004	$0004	AR_DENY_READ
0x0008	$0008	AR_DENY_WRITE
0x0010	$0010	AR_COMPATIBILITY
0x0040	$0040	AR_WRITE_THROUGH
0x0100	$0100	AR_OPEN_COMPRESSED

For Windows, call the **_lread**, **_lwrite**, **_lclose**, and **_llseek** functions.

Calling other functions in Windows returns unexpected results.

NCP Calls

None

See Also

NWConvertFileHandle

_NWConvertHandle (obsolete 6/97)

Converts a NetWare handle to a client platform handle but is now obsolete. Call the **NWConvertHandle** function instead.

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

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )_NWConvertHandle (
   NWCONN_HANDLE          conn,
   nuint8                 accessMode,
   pnuint8                NWHandle,
   nuint32                fileSize,
   NWFILE_HANDLE N_FAR   *fileHandle);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function _NWConvertHandle
  (conn : NWCONN_HANDLE;
   accessMode : nuint8;
   NWHandle : pnuint8;
   fileSize : nuint32;
   Var fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

accessMode

(IN) Specifies the access mode of the file.

NWHandle

(IN) Points to the NetWare handle to be converted.

fileSize

(IN) Specifies the size of the file.

fileHandle

(OUT) Points to the file handle returned.

Return Values

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

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x88 A0	MEMORY_ALLOCATION_ERROR

Remarks

AR_READ and/or AR_WRITE must be used in the *accessMode* parameter. If neither of these are used, **_NWConvertHandle (obsolete 6/97)** sets both.

The following are the access mode definitions:

```
0x0001  AR_READ
0x0002  AR_WRITE
0x0004  AR_DENY_READ
0x0008  AR_DENY_WRITE
0x0010  AR_COMPATIBILITY
0x0040  AR_WRITE_THROUGH
0x0100  AR_OPEN_COMPRESSED
```

NCP Calls

None

NWCreateDirectory

Creates a NetWare directory on the specified server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWCreateDirectory
   (NWCONN_HANDLE   conn,
    NWDIR_HANDLE    dirHandle,
    pnstr8          dirPath,
    nuint8          accessMask);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWCreateDirectory
   (conn : NWCONN_HANDLE;
    dirHandle : NWDIR_HANDLE;
    dirPath : pnstr8;
    accessMask : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle of the root directory for the new directory (0 if the*dirPath* parameter points to the complete path, including the volume name) .

dirPath

(IN) Points to the string containing the name and path of the new directory.

accessMask

(IN) Specifies the access rights mask for the new directory.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8984	NO_CREATE_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899E	INVALID_FILENAME
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	HARDWARE_FAILURE (directory/file already exists)

Remarks

The *accessMask* parameter can be set using one or more of the following:

Hex	Definition
0xFB	TA_ALL
0x01	TA_READ
0x02	TA_WRITE

2	
0x04	TA_OPEN
0x08	TA_CREATE
0x10	TA_DELETE
0x20	TA_OWNERSHIP
0x40	TA_SEARCH
0x80	TA_MODIFY

NOTE: Actual rights are set according to inherited rights.

NCP Calls

0x2222 22 10 Create Directory
0x2222 23 17 Get File Server Information
0x2222 87 01 Open Create File Or Subdirectory

See Also

NWDeleteDirectory

NWDeallocateDirectoryHandle

Deallocates a directory handle allocated by the **NWAllocTemporaryDirectoryHandle** or **NWAllocPermanentDirectoryHandle** function

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWDeallocateDirectoryHandle (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWDeallocateDirectoryHandle
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle to be deallocated.

Return Values

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

0x0000	SUCCESSFUL
0x88	INVALID_CONNECTION

0x8801	INVALID_CONNECTION
0x899B	BAD_DIRECTORY_HANDLE

Remarks

When a workstation terminates or logs out, all directory handles for the workstation are deleted.

NCP Calls

0x2222 22 20 Deallocate Directory Handle

See Also

NWAllocPermanentDirectoryHandle, **NWAllocTempNSDirHandle2**, **NWAllocTemporaryDirectoryHandle**, **NWGetDirectoryHandlePath**

NWDeleteDirectory

Deletes a NetWare directory
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWDeleteDirectory (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          dirPath);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWDeleteDirectory
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   dirPath : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle of the target directory root (0 if the *dirPath* parameter contains the complete path, including the volume name).

dirPath

(IN) Points to the string containing the path (relative to the*dirHandle* parameter) of the directory being deleted.

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898A	NO_DELETE_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899F	DIRECTORY_ACTIVE
0x89A0	DIRECTORY_NOT_EMPTY
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	Failure

NCP Calls

0x2222 22 11 Delete Directory
0x2222 23 17 Get File Server Information
0x2222 87 08 Delete A File Or Subdirectory

See Also

NWCreateDirectory

NWDeleteTrustee

Removes a trustee from the specified directory or a trustee list for a file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWDeleteTrustee (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          dirPath,
   nuint32         objID);
```

Pascal Syntax

```
#include <nwdentry.inc>

Function NWDeleteTrustee
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   dirPath : pnstr8;
   objID : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle for the directory whose trustee list is being deleted (0 if the*dirPath* parameter points to the complete path, including the volume name).

dirPath

(IN) Points to the directory from which the trustee is being removed.

objID

(IN) Specifies the bindery object ID for the trustee being deleted.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	N0_MODIFY_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x8999	DIRECTORY_FULL
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FC	NO_SUCH_OBJECT
0x89FD	BAD_STATION_NUMBER
0x89FE	TRUSTEE_NOT_FOUND
0x89FF	HARDWARE_FAILURE, Failure

Remarks

NWDeleteTrustee also revokes the rights of the trustee in the specified directory.

To delete a trustee, the requesting workstation must have access control rights for 3.x servers or parental rights for 2.x servers in the directory or in a parent directory.

Deleting the explicit assignment of an trustee object in a directory is not the same as assigning no rights to the object in the directory. If no rights are assigned in a directory, the object inherits the same rights as the

parent directory.

NCP Calls

0x2222 22 14 Delete Trustee From Directory
0x2222 22 43 Trustee Remove Ext
0x2222 23 17 Get File Server Information
0x2222 87 11 Delete Trustee Set From File Or Subdirectory

See Also

NWAddTrustee, NWIntScanForTrustees, NWParseNetWarePath

NWDeleteTrusteeFromDirectory

Removes a trustee from a directory trustee list
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWDeleteTrusteeFromDirectory (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint32         objID);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWDeleteTrusteeFromDirectory
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   objID : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle for the directory whose trustee list is being modified (zero if the *path* parameter points to the complete path, including the volume name).

path

(IN) Points to an absolute path (or a path relative to the *dirHandle* parameter) specifying the directory from which the trustee is being removed.

objID

(IN) Specifies the bindery object ID for the trustee being deleted.

Return Values

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

0x0000	SUCCESSFUL

Remarks

NWDeleteTrusteeFromDirectory revokes the rights for a trustee in a specific directory. The requesting workstation must have parental rights in the directory, or in a parent directory, to delete a trustee.

Deleting the explicit assignment of an trustee object in a directory is not the same as assigning no rights to the object in the directory. If no rights are assigned in a directory, the object inherits the same rights it has in the parent directory.

NCP Calls

0x2222 22 14 Delete Trustee From Directory
0x2222 22 43 Trustee Remove Ext
0x2222 23 17 Get File Server Information
0x2222 87 11 Delete Trustee Set From File Or Subdirectory

See Also

NWAddTrusteeToDirectory, **NWParseNetWarePath**, **NWScanDirectoryForTrustees2**

NWEraseFiles (obsolete 6/96)

Deletes NetWare files from the server but is now obsolete. Call the **NWIntEraseFiles** function instead.

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWEraseFiles
   (NWCONN_HANDLE    conn,
    NWDIR_HANDLE     dirHandle,
    pnstr8           path,
    nuint8           searchAttrs);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWEraseFiles
   (conn : NWCONN_HANDLE;
    dirHandle : NWDIR_HANDLE;
    path : pnstr8;
    searchAttrs : nuint8;
    augmentFlag : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle containing the file to erase.

dirHandle

(IN) Specifies the directory handle of the file to be erased (0 if the*path* parameter contains the complete path, including the volume name).

path

(IN) Points to the string containing the file path, including the file name, of the file to be erased.

searchAttrs

(IN) Specifies the search attributes.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

The *searchAttrs* parameter includes system and/or hidden files. If only the system bit is set in the *searchAttrs* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files that are not designated either hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

Search attributes to use in finding a file follow:

```
0x00   none
0x02   FA_HIDDEN
0x04   FA_SYSTEM
0x06   both
```

The *path* parameter can specify either the complete path name for a file or a path relative to the current working directory. For example, if the complete path name is SYS:ACCOUNT/DOMEST/TARGET.DAT and the directory handle mapping is SYS:ACCOUNT, the value of the *path* parameter could be either of the following:

```
SYS:ACCOUNT/DOMEST/TARGET.DAT or
DOMEST/TARGET.DAT
```

The *path* parameter can point to wildcards in the file name only. Wildcard matching uses the method defined by the application when it passes a

wildcard character.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 68 Erase File
0x2222 87 08 Delete A File Or Subdirectory

See Also

NWPurgeErasedFiles, **NWRenameFile**

NWFileSearchContinue (obsolete 6/96)

Iteratively retrieves all directory entries matching the *searchPath* parameter but is now obsolete. Call the **NWIntFileSearchContinue** function instead.

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWFileSearchContinue
   (NWCONN_HANDLE    conn,
    nuint8           volNum,
    nuint16          dirID,
    nuint16          searchContext,
    nuint8           searchAttr,
    pnstr8           searchPath,
    pnuint8          retBuf);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWFileSearchContinue
   (conn : NWCONN_HANDLE;
    volNum : nuint8;
    dirID : nuint16;
    searchContext : nuint16;
    searchAttr : nuint8;
    searchPath : pnstr8;
    retBuf : pnuint8;
    augmentFlag : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number returned by the initialize function.

dirID

(IN) Specifies the directory ID returned by the initialize function.

searchContext

(IN) Specifies the sequence number returned by the **NWIntFileSearchInitialize** function.

searchAttr

(IN) Specifies the attributes to apply to the search.

searchPath

(IN) Points to the path (file name, directory name, or wildcard).

retBuf

(OUT) Points to the information returned by the server.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST

Remarks

On the first iteration, use the sequence number returned by the **NWIntFileSearchInitialize** function. For subsequent iterations, use the *sequenceNumber* field from the SEARCH_FILE_INFO or SEARCH_DIR_INFO structure.

NWFileSearchContinue (obsolete 6/96) returns two different search structures depending on whether the match is a directory or a file. The application is responsible for determining the type of match, or for limiting the search to files or directories only. The two search structures are SEARCH_FILE_INFO and SEARCH_DIR_INFO.

Search attributes follow:

C Value	**Pascal Value**	**Value Name**

0x00	$00	FA_NORMAL
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x10	$10	FA_DIRECTORY

If other values are used for search attributes, each will be treated as FA_NORMAL.

NCP Calls

0x2222 63 File Search Continue

See Also

NWIntFileSearchInitialize

NWFileSearchInitialize (obsolete 6/97)

Searches for files on a server but is now obsolete. Call the **NWIntFileSearchInitialize** function instead.

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWFileSearchInitialize (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   pnuint8         volNum,
   pnuint16        dirID,
   pnuint16        iterHnd,
   pnuint8         accessRights);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWFileSearchInitialize
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   volNum : pnuint8;
   dirID : pnuint16;
   iterhandle : pnuint16;
   accessRights : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the base directory handle to search.

path

(IN) Points to the path (relative to the *dirHandle* parameter) on which to initialize the search.

volNum

(OUT) Points to the corresponding volume number.

dirID

(OUT) Points to the directory ID corresponding to the specified path.

iterHnd

(OUT) Points to a sequence number to be used in calling the **NWIntFileSearchContinue** function (initially -1).

accessRights

(OUT) Points to the access rights to the specified directory.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

If the directory handle is unknown, a value of 0 should be passed. In the absence of the directory handle, the *path* parameter needs to specify the volume as well.

NCP Calls

0x2222 62 File Search Initialize

See Also

NWIntFileSearchContinue

NWFileServerFileCopy

Copies a file or portion of a file from a source to a destination on the same NetWare server

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWFileServerFileCopy (
   NWFILE_HANDLE   srcFileHandle,
   NWFILE_HANDLE   dstFileHandle,
   nuint32         srcOffset,
   nuint32         dstOffset,
   nuint32         bytesToCopy,
   pnuint32        bytesCopied);
```

Pascal Syntax

```
#include<nwfile.inc>

Function NWFileServerFileCopy
  (srcFileHandle : NWFILE_HANDLE;
   dstFileHandle : NWFILE_HANDLE;
   srcOffset : nuint32;
   dstOffset : nuint32;
   bytesToCopy : nuint32;
   bytesCopied : pnuint32
) : NWCCODE;
```

Parameters

srcFileHandle

(IN) Specifies the source file handle (index).

dstFileHandle

(IN) Specifies the destination file handle (index).

srcOffset

(IN) Specifies the offset in the source file where the copying is to begin.

dstOffset

(IN) Specifies the offset in the destination file where the copying is to begin.

bytesToCopy

(IN) Specifies the maximum number of bytes to copy.

bytesCopied

(OUT) Points to the number of bytes actually copied, or the size of a new destination file (optional).

Return Values

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

0x0000	SUCCESSFUL
0x0006	Invalid File Handle
0x8830	NOT_SAME_CONNECTION
0x8901	ERR_INSUFFICIENT_SPACE
0x8983	IO_ERROR_NETWORK_DISK
0x8988	INVALID_FILE_HANDLE
0x8993	NO_READ_PRIVILEGES
0x8994	NO_WRITE_PRIVILEGES_OR_READONLY
0x8995	FILE_DETACHED
0x8996	SERVER_OUT_OF_MEMORY
0x89A2	READ_FILE_WITH_RECORD_LOCKED

Remarks

NWFileServerFileCopy is very efficient since the data does not come to the workstation; the server handles the duplication of the data internally.

If the source and destination files do not reside on the same server,

NOT_SAME_CONNECTION is returned.

You must pass OS file handles in the *srcFileHandle* and *dstFileHandle* parameters. Use the appropriate OS functions that create and open files to return the file handles, depending on whether the destination file is a new or an existing file.

If the destination file is new, the *bytesCopied* parameter points to the size of the destination file. Otherwise, it points to the number of bytes copied.

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

Under OS/2, the given handles are converted to NetWare handles via the NetWare IFS.

Under DOS, **NWFileServerFileCopy** is a direct call to the shell. The shell handles the copy.

NCP Calls

0x2222 74 Copy From One File To Another

NWGetCompressedFileLengths

Returns information about the lengths of a compressed file
Local Servers: blocking
Remote Servers: blocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwfile.h>

int NWGetCompressedFileLengths (
   int    handle,
   LONG  *uncompressedLength,
   LONG  *compressedLength;
```

Parameters

handle

(IN) Specifies the file handle for which to return the lengths.

uncompressedLength

(OUT) Points to the length of the file in an uncompressed state.

compressedLength

(OUT) Points to the length of the file after being compressed.

Return Values

0	Success
0xFF	Failure

Remarks

NWGetCompressedFileLengths returns information about the lengths of a compressed file.

If the *handle* parameter represents a file that is not compressed, the lengths will be invalid.

The *uncompressedLength* parameter specifies the length normally seen in

directory listings.

See Also

NWSetCompressedFileLengths

NWGetDirectoryEntryNumber

Returns file information for a specified file under DOS and the name space associated with the specified directory handle

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY (NWCCODE) NWGetFileDirEntryNumber (
   NWCONN_HANDLE   conn,
   nuint8          dirHandle,
   pnuint32        volumeNum,
   pnuint32        directoryEntry,
   pnuint32        DOSDirectoryEntry,
   pnuint32        nameSpace,
   pnuint32        parentDirEntry,
   pnuint32        parentDOSDirEntry);
```

Pascal Syntax

```
Function NWGetFileDirEntryNumber
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   volumeNum : pnuint32;
   directoryEntry : pnuint32;
   DOSDirectoryEntry : pnuint32;
   nameSpace : pnuint32;
   parentDirEntry : pnuint32;
   parentDOSDirEntry : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server conneciton handle.

dirHandle

(IN) Specifies the one byte directory handle.

volumeNum

(OUT) Points to the volume number of the directory handle.

directoryEntry

(OUT) Points to the directory entry number in the name space associated with the *dirHandle* parameter.

DOSDirectoryEntry

(OUT) Points to the directory entry number in the DOS name space.

nameSpace

(OUT) Points to the name space associated with the *directoryEntry* and *parentDirEntry* parameters.

parentDirEntry

(OUT) Points to the parent directory entry number of the directory handle in the name space associated with the *dirHandle* parameter.

parentDOSDirEntry

(OUT) Points to the parent directory entry number of the directory handle in the DOS name space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

NWGetDirectoryEntryNumber returns the volume number, directory entry numbers, parent directory entry numbers in the DOS name space, and the name space associated with the directory handle.

One way to create the directory handle is to call the **NWAllocTempNSDirHandle2** function. If you specify a long directory name, the created directory handle will be associated with the LONG name space. If a DOS directory name is specified, the created directory handle will be associated with the DOS name space.

The *nameSpace* parameter can have the following values:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_LONG

NCP Calls

87 31 Get File Information

See Also

NWAllocTempNSDirHandle2

NWGetDirectoryHandlePath

Returns the path name of the directory associated with the given directory handle

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetDirectoryHandlePath (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          dirPath);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWGetDirectoryHandlePath
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   dirPath : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle for the directory whose path is to be reported.

dirPath

(OUT) Points to the directory path name associated with the *dirHandle* parameter.

Return Values

These are common return values; see Return Values for more

information.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 9B	BAD_DIRECTORY_HANDLE
0x89 A1	DIRECTORY_IO_ERROR

Remarks

NWGetDirectoryHandlePath allows a client to retrieve the full directory path of the directory indexed by the *dirHandle* parameter. The string accessed by the *dirPath* parameter contains a path name in the following format:

```
Volume Name:Directory\Subdirectory\....
```

The string accessed by the *dirPath* parameter does not contain the name of the server. Its maximum length is 255 bytes.

Under NETX, if an invalid connection handle is passed to the *conn* parameter, **NWGetDirectoryHandlePath** will return 0x0000. An error will never be returned by NETX since NETX always chooses a default connection handle if the connection handle cannot be resolved.

NETX tries to resolve the connection ID through the preferred server first. If a preferred server does not exist, the request is directed to the default server (or the server implied by the default drive). If the default drive is mapped to a local drive, the shell directs the request to the primary server as the lowest connection priority.

NCP Calls

0x2222 22 01 Get Directory Path

See Also

NWAllocTemporaryDirectoryHandle, **NWDeallocateDirectoryHandle**

NWGetDirSpaceInfo

Returns information on space usage for a volume
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetDirSpaceInfo (
   NWCONN_HANDLE          conn,
   NWDIR_HANDLE           dirHandle,
   nuint16                volNum,
   DIR_SPACE_INFO N_FAR  *spaceInfo);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWGetDirSpaceInfo
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   volNum : nuint16;
   Var spaceInfo : DIR_SPACE_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle (nuint16).

dirHandle

(IN) Specifies the directory handle associated with the desired directory path (0 if volume information is to be returned).

volNum

(IN) Specifies the volume number to return space information for (0 if directory information is to be returned).

spaceInfo

(OUT) Points to the DIR_SPACE_INFO structure.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

If the *dirHandle* parameter is zero, **NWGetDirSpaceInfo** returns the volume information to the DIR_SPACE_INFO structure.

The *purgeableBlocks* parameter is set to 0 if the *dirHandle* parameter contains a nonzero value.

The *availableBlocks* field is the only field that returns information when disk space restrictions are in effect. Tthe rest of the structure fields contain volume wide information. If disk space restrictions are not in effect, the *availableBlocks* field will contain the number of blocks available for use on the entire volume.

NCP Calls

0x2222 22 44 Get Volume Purge Information
0x2222 22 45 Get Dir Info

NWGetDirSpaceLimitList

Determines the actual space limitations for a directory
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetDirSpaceLimitList (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnuint8         returnBuf);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWGetDirSpaceLimitList
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   returnBuf : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle pointing to the desired directory.

returnBuf

(OUT) Points to a 512-byte buffer containing the returned space list.

Return Values

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

0x0000	SUCCESSFUL

Remarks

To find the actual amount of space available to a directory, scan all of the *current* fields and use the smallest one. If no entries are returned, no space restrictions exist for the specified directory.

NOTE: All restrictions are returned in units of 4K blocks.

The NW_LIMIT_LIST structure will be used to return space list information. The NW_LIMIT_LIST structure is pointed to by the *returnBuf* parameter.

NCP Calls

None

NWGetDiskIOsPending

Returns the number of pending disk IOs the file server has at the specified point in time

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x

Platform: NLM

SMP Aware: No

Service: File System

Syntax

```
#include <nwfile.h>

int NWGetDiskIOsPending (
   void);
```

Return Values

Returns the number of pending disk IOs the file server has upon successful completion.

Remarks

The value returned by **NWGetDiskIOsPending** is the same as the value for "Current disk requests" as reported by the MONITOR.NLM file.

NWGetEffectiveRights

Returns effective rights for the specified directory
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetEffectiveRights (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   pnuint16        effectiveRights);
```

Pascal Syntax

```
#include <nwdentry.inc>

Function NWGetEffectiveRights
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   effectiveRights : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle associated with the directory path for which the effective rights are desired (0 if the *path* parameter points to the complete path, including the volume name).

path

(IN) Points to the absolute path (or a path relative to the *dirHandle* parameter) of the directory whose effective rights mask is being returned.

effectiveRights

(OUT) Points to the effective rights mask for the directory.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	Failure

Remarks

To determine the effective rights of the requesting workstation, **NWGetEffectiveRights** performs a logical AND between the maximum rights mask of the directory and the current trustee rights of the workstation.

The current trustee rights are obtained by performing a logical OR between a trustee access mask and the trustee access mask of any object to which the process is security equivalent.

The current trustee rights can be explicitly listed in the directory or inherited from the parent directory. The maximum rights masks of parent directories do not affect inherited trustee rights.

The *effectiveRights* parameter returned to the client indicates which of the eight possible directory rights the client has in the targeted directory. An *effectiveRights* parameter of zero indicates the client has no rights in the target directory.

The maximum rights mask bits are defined in the table below:

C Value	Pascal Value	Value Description
0x0001	$0001	TR_READ
0x0002	$0002	TR_WRITE
0x0008	$0008	TR_CREATE
0x0010	$0010	TR_DELETE
0x0010	$0020	TR_OWNERSHIP
0x0040	$0040	TR_FILE_SCAN
0x0080	$0080	TR_MODIFY

For 3.x-4.x servers, **NWGetEffectiveRights** works on files as well as directories. For 2.x servers, **NWGetEffectiveRights** works only on paths.

See Displaying effective rights: Example.

NCP Calls

0x2222 22 3 Get Effective Directory Rights
0x2222 22 42 Get Effective Rights
0x2222 23 17 Get File Server Information
0x2222 87 29 Get Effective Directory Rights

NWGetExtendedFileAttributes2

Returns the NetWare extended file attributes for the specified file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetExtendedFileAttributes2 (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   pnuint8         extAttrs);
```

Pascal Syntax

```
#include<nwfile.inc>

Function NWGetExtendedFileAttributes2
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   extAttrs : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle of the new root directory.

path

(IN) Points to the string containing the name and path of the new directory.

extAttrs

(OUT) Points to the extended attributes of the file.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE
0x8989	NO_SEARCH_PRIVILEGES
0x8993	NO_READ_RRIVILEGES
0x8994	NO_WRITE_PRIVILEGES_OR_READONLY
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	NO_FILES_FOUND_ERROR

Remarks

NWGetExtendedFileAttributes2 requires Search rights to the directory where the file resides.

The *path* parameter can specify the complete path name or a path relative to the current working directory. For example, if the complete path name is SYS:ACCOUNT/DOMEST/TARGET.DAT and the directory handle mapping is SYS:ACCOUNT, the *path* parameter could be the following:

```
SYS:ACCOUNT/DOMEST/TARGET.DAT or
DOMEST/TARGET.DAT
```

The information accessed by the *extAttrs* parameter is interpreted as follows:

Bits	Function
0-2	Search mode bits
4	Transaction bit
5	Index bit
6	Read audit bit
7	Write audit bit

NCP Calls

0x2222 23 15 Scan File Information

See Also

NWSetExtendedFileAttributes2

NWGetFileConnectionID

Returns the connection handle of the server owning the specified file handle
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetFileConnectionID (
   NWFILE_HANDLE          fileHandle,
   NWCONN_HANDLE N_FAR  *conn);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWGetFileConnectionID
  (fileHandle : NWFILE_HANDLE;
   Var conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

fileHandle

(IN) Specifies the file handle.

conn

(OUT) Points to the connection handle.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x89FD	UNKNOWN_REQUEST

Remarks

The server connection handle identifies a specific NetWare server to workstation connection.

NWGetFileConnectionID only works with VLMs loaded; it will not work with NETX. If NETX is loaded, UNKNOWN_REQUEST will be returned.

NCP Calls

None

NWGetFileDirEntryNumber

Returns file information for a specified file under DOS and the name space associated with the specified file handle

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY (NWCCODE) NWGetFileDirEntryNumber (
   NWFILE_HANDLE   fileHandle,
   pnuint32        volumeNum,
   pnuint32        directoryEntry,
   pnuint32        DOSDirectoryEntry,
   pnuint32        nameSpace,
   pnuint32        dataStream,
   pnuint32        parentDirEntry,
   pnuint32        parentDOSDirEntry);
```

Pascal Syntax

```
Function NWGetFileDirEntryNumber
  (fileHandle : NWFILE_HANDLE;
   volumeNum : pnuint32;
   directoryEntry : pnuint32;
   DOSDirectoryEntry : pnuint32;
   nameSpace : pnuint32;
   dataStream : pnuint32;
   parentDirEntry : pnuint32;
   parentDOSDirEntry : pnuint32
) : NWCCODE;
```

Parameters

fileHandle

(IN) Specifies the file handle.

volumeNum

(OUT) Points to the volume number of the file handle.

directoryEntry

(OUT) Points to the directory entry number in the name space associated with the *fileHandle* parameter.

DOSDirectoryEntry

(OUT) Points to the directory entry number in the DOS name space.

nameSpace

(OUT) Points to the name space associated with the *directoryEntry* and *parentDirEntry* parameters.

dataStream

(OUT) Points to the data stream number if the name space is NW_NS_MAC:

1 Data fork

0 Resource fork and anything else

parentDirEntry

(OUT) Points to the parent directory entry number of the file handle in the name space associated with the *fileHandle* parameter.

parentDOSDirEntry

(OUT) Points to the parent directory entry number of the file handle in the DOS name space.

Return Values

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

0x0000	SUCCESSFUL
0x0006	INVALID_HANDLE
0x8801	INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE

Remarks

NWGetFileDirEntryNumber returns the volume number, directory entry numbers, parent directory entry numbers in the DOS name space, and the name space associated with the file handle.

One way to create the file handle is to call the **NWOpenNSEntry** function. If you specify a long file name, the created file handle will be associated with the LONG name space. If a DOS file name is specified, the created file handle will be associated with the DOS name space.

The *nameSpace* parameter can have the following values:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_LONG

NCP Calls

87 31 Get File Information

See Also

NWOpenNSEntry

NWGetSparseFileBitMap

Returns a bit map showing which blocks in a sparse file contain data
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY (NWCCODE)NWGetSparseFileBitMap (
   NWCONN_HANDLE   conn,
   nuint32         fileHandle,
   nint16          flag,
   nuint32         offset,
   pnuint32        blockSize,
   pnuint8         bitMap);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWGetSparseFileBitMap
  (conn : NWCONN_HANDLE;
   fileHandle : NWFILE_HANDLE;
   flag : nint16;
   offset : nuint32;
   blockSize : pnuint32;
   bitMap : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fileHandle

(IN) Specifies the 4-byte OS or NetWare file handle. If a NetWare file handle is used, a connection handle must be passed.

flag

(IN) Specifies whether the *fileHandle* parameter contains an OS or NetWare handle.

offset

(IN) Specifies the starting offset of the bit map in bytes.

blockSize

(OUT) Points to the size of the allocation block.

bitMap

(OUT) Points to a 512-byte array to receive the bit map (1 bit for each block).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE

Remarks

NWGetSparseFileBitMap contains one bit for each block in the sparse file. A one (1) indicates there is data in the block; a zero (0) indicates there isn't any data in the block.

Use the *conn* parameter when NETX is running or the *fileHandle* parameter contains a NetWare handle (otherwise ignored).

If the *flag* parameter is 0, the *fileHandle* parameter contains an OS (DOS or OS/2) file handle. If the *flag* parameter is nonzero, the *fileHandle* parameter contains a 4-byte NetWare handle.

The *bitMap* parameter must point to an array of 512 bytes.

NCP Calls

0x2222 85 Get Sparse File Data Block Bit Map

NWIntEraseFiles

Deletes NetWare files from the server
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWIntEraseFiles (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint8          searchAttrs,
   nuint16         augmentFlag);
```

Parameters

conn

(IN) Specifies the NetWare server connection handle containing the file to erase.

dirHandle

(IN) Specifies the directory handle of the file to be erased (0 if the*path* parameter contains the complete path including the volume name).

path

(IN) Points to the string containing the file path (including the file name) of the file to be erased.

searchAttrs

(IN) Specifies the search attributes.

augmentFlag

(IN) Specifies if wildcards are augmented:

0 = wildcards are not augmented
nonzero = wildcards are augmented

Return Values

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

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 8A	NO_DELETE_PRIVILEGES
0x89 8D	SOME_FILES_AFFECTED_IN_USE
0x89 8E	NO_FILES_AFFECTED_IN_USE
0x89 8F	SOME_FILES_AFFECTED_READ_ONLY
0x89 90	NO_FILES_AFFECTED_READ_ONLY
0x89 98	VOLUME_DOES_NOT_EXIST
0x89 9B	BAD_DIRECTORY_HANDLE
0x89 9C	INVALID_PATH
0x89 FF	NO_FILES_FOUND_ERROR

Remarks

The *searchAttrs* parameter includes system and/or hidden files. If only the system bit is set in the *searchAttrs* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files that are not designated either hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

Search attributes to use in finding a file follow:

```
0x00   none
0x02   FA_HIDDEN
0x04   FA_SYSTEM
0x06   both
```

The *path* parameter can specify either a complete path name or a path relative to the current working directory. For example, if the complete path name is SYS:ACCOUNT/DOMEST/TARGET.DAT and the directory handle mapping is SYS:ACCOUNT, the value of the *path* parameter could be either of the following:

```
SYS:ACCOUNT/DOMEST/TARGET.DAT or DOMEST/TARGET.DAT
```

The *path* parameter can point to wildcards in the file name only. Wildcard matching uses the method defined by the application when it passes a wildcard character.

The client must have file deletion privileges in the target directory or **NWIntEraseFiles** will fail.

If a file has the immediate purge attribute set, the file cannot be recovered.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 68 Erase File
0x2222 87 08 Delete A File Or Subdirectory

See Also

NWPurgeDeletedFile, **NWPurgeErasedFiles**, **NWRecoverDeletedFile**, **NWRenameFile**

NWIntFileSearchContinue

Iteratively retrieves all directory entries matching the *searchPath* parameter in the DOS name space

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWIntFileSearchContinue (
   NWCONN_HANDLE   conn,
   nuint8          volNum,
   nuint16         dirID,
   nuint16         searchContext,
   nuint8          searchAttr,
   pnstr8          searchPath,
   pnuint8         retBuf,
   nuint16         augmentFlag);
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number returned by the initialize function.

dirID

(IN) Specifies the directory ID returned by the initialize function.

searchContext

(IN) Specifies the sequence number returned by the **NWIntFileSearchInitialize** function.

searchAttr

(IN) Specifies the attributes to apply to the search.

searchPath

(IN) Points to the path (file name, directory name, or wildcard).

retBuf

(OUT) Points to the information returned by the server.

augmentFlag

(IN) Specifies if wildcards are augmented:

0 = wildcards are not augmented
nonzero = wildcards are augmented

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x89FF	NO_FILES_FOUND_ERROR

Remarks

NWIntFileSearchContinue returns two different search structures depending on whether the match is a directory or a file. The application is responsible for determining the type of match, or for limiting the search to files or directories only. The two search structures are SEARCH_FILE_INFO and SEARCH_DIR_INFO.

On the first iteration, use the sequence number returned by the **NWIntFileSearchInitialize** function. For subsequent iterations, use the *iterHnd* field from the SEARCH_FILE_INFO or SEARCH_DIR_INFO structure.

Valid search attributes follow:

C Value	**Pascal Value**	**Value Name**
0x00	$00	FA_NORMAL
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x10	$10	FA_DIRECTORY

If other values are used for search attributes, each will be treated as

FA_NORMAL.

NCP Calls

0x2222 63 File Search Continue

NWIntFileSearchInitialize

Searches for files on a server
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWIntFileSearchInitialize (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   pnuint8         volNum,
   pnuint16        dirID,
   pnuint16        iterHnd,
   pnuint8         accessRights,
   nuint16         augmentFlag);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWIntFileSearchInitialize
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   volNum : pnuint8;
   dirID : pnuint16;
   iterhandle : pnuint16;
   accessRights : pnuint8;
   augmentFlag : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the base directory handle to search.

path

(IN) Points to the path (relative to the*dirHandle* parameter) on which to

initialize the search.

volNum

(OUT) Points to the corresponding volume number.

dirID

(OUT) Points to the directory ID corresponding to the specified path.

iterHnd

(OUT) Points to a sequence number to be used in calling the **NWIntFileSearchContinue** function (initially -1).

accessRights

(OUT) Points to the access rights of the workstation to the specified directory.

augmentFlag

(IN) Is reserved (pass in zero).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

A value of 0 should be passed to the *dirHandle* parameter if the directory handle is not known. In the absence of the directory handle, the *path* parameter needs to specify the volume as well.

NCP Calls

0x2222 62 File Search Initialize

See Also

NWIntFileSearchContinue

NWIntMoveDirEntry

Moves or renames a directory entry (file or directory) on the same server (same volume) in the DOS name space

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWIntMoveDirEntry (
   NWCONN_HANDLE   conn,
   nuint8          searchAttrs,
   NWDIR_HANDLE    srcDirHandle,
   pnstr8          srcPath,
   NWDIR_HANDLE    dstDirHandle,
   pnstr8          dstPath,
   nuint16         augmentFlag);
```

Pascal Syntax

```
#include <nwdentry.inc>

Function NWIntMoveDirEntry
  (conn : NWCONN_HANDLE;
   searchAttrs : nuint8;
   srcDirHandle : NWDIR_HANDLE;
   srcPath : pnstr8;
   dstDirHandle : NWDIR_HANDLE;
   dstPath : pnstr8;
   augmentFlag : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

searchAttrs

(IN) Specifies the attributes to use in searching for the source entries.

srcDirHandle

(IN) Specifies the directory handle for the source directory (not optional, cannot be zero).

srcPath

(IN) Points to the source path (wildcards are allowed).

dstDirHandle

(IN) Specifies the NetWare directory handle for the destination directory.

dstPath

(IN) Points to the path name to use for the destination entry.

augmentFlag

(IN) Specifies if wildcards are augmented:

0 = wildcards are not augmented
nonzero = wildcards are augmented

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8987	WILD_CARDS_IN_CREATE_FILE_NAME
0x898B	NO_RENAME_PRIVILEGES
0x898D	SOME_FILES_AFFECTED_IN_USE
0x898E	NO_FILES_AFFECTED_IN_USE "All files in use"
0x898F	SOME_FILES_AFFECTED_READ_ONLY
0x8990	NO_FILES_AFFECTED_READ_ONLY "Read-only access to volume"
0x8991	SOME_FILES_RENAMED_NAME_EXISTS
0x8992	NO_FILES_RENAMED_NAME_EXISTS
0x899A	RENAMING_ACROSS_VOLUMES
0x899B	BAD_DIRECTORY_HANDLE

0x899C	INVALID_PATH
0x89A4	ERR_RENAME_DIR_INVALID
0x89FF	NO_FILES_FOUND_ERROR

Remarks

To call **NWIntMoveDirEntry**, you must have file modification privileges in both the source and the target directories.

The specified paths are relative to the specified directory handles. NetWare 3.11 and above accepts paths relative to the directory handle, as well as full paths that include the volume. If full names are used, be careful that the maximum request length is not exceeded. Path names larger than 255 are not supported.

The *searchAttrs* parameter specifies the kind of entry to look for (hidden, system, etc.). If only the system bit is set, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files that are not designated either hidden or system are affected.

For NetWare 2.x, **NWIntMoveDirEntry** only works on files.

The *searchAttrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The advantage of calling **NWIntMoveDirEntry** over DOS, OS/2, or other

functions is its speed and efficiency. Since the move is within the server, the entry in the file system is simply deleted from the source and inserted in the destination. Moving directory entries occurs only on the file system level. There is no physical transfer of data between the source and the destination.

NOTE: **NWIntMoveDirEntry** will move files within the same volume only. If you attempt to move a file across different volumes, RENAMING_ACROSS_VOLUMES will be returned.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 69 Rename File
0x2222 87 04 Rename Or Move A File Or Subdirectory

NWIntScanDirectoryInformation2

Returns directory information for a directory specified by the connection handle, directory handle, and directory path

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWIntScanDirectoryInformation2 (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          srchPath,
   pnuint8         sequence,
   pnstr8          dirName,
   pnuint32        dirDateTime,
   pnuint32        ownerID,
   pnuint8         rightsMask,
   nuint16         augmentFlag);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWIntScanDirectoryInformation2
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   searchPath : pnstr8;
   sequence : pnuint8;
   dirName : pnstr8;
   dirDateTime : pnuint32;
   ownerID : pnuint32;
   rightsMask : pnuint8;
   augmentFlag : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle for the directory being

scanned.

srchPath

(IN) Points to an absolute directory path with a maximum length of 255 (or a path relative to the directory handle) and a search pattern (optional).

sequence

(IN/OUT) Points to a 9-byte sequence number to be used for subsequent calls (the first 4 bytes should be 0xFF initially).

dirName

(OUT) Points to the directory name found (256 bytes, optional).

dirDateTime

(OUT) Points to the creation date and time of the directory (4 bytes, optional) in the DOS date and time format.

ownerID

(OUT) Points to the object ID of the owner for the directory (optional).

rightsMask

(OUT) Points to the maximum rights mask for the directory found (optional).

augmentFlag

(IN) Specifies if wildcards are augmented:

0 = wildcards are not augmented

nonzero = wildcards are augmented

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89FF	NO_FILES_FOUND_ERROR

Remarks

All parameter fields must be filled. However, NULL may be substituted in parameters where no information is desired.

The *dirHandle* parameter can be zero if the *srchPath* parameter points to the complete path, including the volume name.

The string accessed by the *srchPath* parameter can include wildcard characters. If wildcards are used, only the directory information for the first matching directory is returned.

The *dirName* parameter is invalid when getting volume information for NetWare 2.2 servers.

The *dirDateTime* parameter does not point to valid information on 2.x servers and does not point to valid information for the root directory (volume) on 2.2 servers. Set the *dirDateTime* parameter to zero when getting volume information for 2.2 servers. For volumes, 2.x servers do not return a directory name, and the *dirDateTime* parameter always points to zero.

The *rightsMask* parameter can have the following values:

```
0x00 = TA_NONE
0x01 = TA_READ
0x02 = TA_WRITE
0x04 = TA_OPEN
0x08 = TA_CREATE
0x10 = TA_DELETE
0x20 = TA_OWNERSHIP
0x40 = TA_SEARCH
0x80 = TA_MODIFY
0xFB = TA_ALL
```

NOTE: TA_OPEN is obsolete in NetWare 3.x and above.

NCP Calls

0x2222 22 01 Get Directory Path
0x2222 22 02 Scan Directory Information
0x2222 23 17 Get File Server Information
0x2222 87 02 Initialize Search
0x2222 87 03 Search For File Or Subdirectory
0x2222 87 06 Obtain File Or Subdirectory Information

See Also

NWParseNetWarePath

NWIntScanDirEntryInfo

Obtains information about NetWare 3.x and 4.x directory entries (files or directories) in the DOS name space

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWIntScanDirEntryInfo (
   NWCONN_HANDLE         conn,
   NWDIR_HANDLE          dirHandle,
   nuint16               attrs,
   pnuint32              iterHandle,
   pnuint8               searchPattern,
   NWENTRY_INFO N_FAR   *entryInfo,
   nuint16              augmentFlag);
```

Pascal Syntax

```
#include <nwdentry.inc>

Function NWIntScanDirEntryInfo
  (conn :  NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   attrs : nuint16;
   iterHandle : pnuint32;
   searchPattern : pnuint8;
   Var entryInfo : NWENTRY_INFO;
   augmentFlag : nuint16
) : NWCCODE ;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare DOS directory handle indexing the directory to scan (not optional, cannot be 0).

attrs

(IN) Specifies the attributes to be used for the scan.

iterHandle

(IN/OUT) Points to an nuint32 buffer to receive the search sequence from the server.

searchPattern

(IN) Points to the name of the entry for which to scan (wildcards are allowed).

entryInfo

(OUT) Points to the NWENTRY_INFO structure (zeroed out initially).

augmentFlag

(IN) Specifies if wildcards are augmented:

0 = wildcards are not augmented
nonzero = wildcards are augmented

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8989	NO_SEARCH_PRIVILEGES
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89FF	NO_FILES_FOUND_ERROR

Remarks

NWIntScanDirEntryInfo can only be called with non-augmented wildcards if the *augmentFlag* parameter is set to 0. For example, *.* will match anything with a period, while * will match any string.

NWIntScanDirEntryInfo will support augmented wildcard characters if the *augmentFlag* parameter is set to 1 or if the high-order bits have been manually set. For example, * will now match zero or more characters up

to a period or an end-of-string. See the explanation under "wildcard characters" in the Glossary.

On the first call, the *iterHandle* parameter should point to 0xFFFFFFFF. After that, the server manages the information. All scanning is complete when the server returns 0x89FF.

The *searchPattern* parameter cannot point to any path elements and the *dirHandle* parameter must index the complete path.

NWIntScanDirEntryInfo can also be used to scan for information about other directories, including the root directory. In this mode, the *dirHandle* parameter needs to index the root or a directory, and the *searchPattern* parameter needs to point to NULL.

NWIntScanDirEntryInfo works with the DOS name space only. Path and file names must be upper cased. To scan using alternate name spaces, convert the path to a DOS name space by calling either the **NWGetNSPath** or **NWScanNSEntryInfo** function. You can also scan the Macintosh name space by calling the **NWAFPScanFileInformation** function.

The *attrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x10	$10	FA_DIRECTORY

The NWENTRY_INFO structure should be initialized to 0 before **NWIntScanDirEntryInfo** is called for the first time.

NCP Calls

0x2222 22 01 Get Directory Path
0x2222 22 30 Scan A Directory
0x2222 22 31 Get Directory Entry

See Also

NWAFPScanFileInformation, **NWGetNSInfo**, **NWIntScanExtendedInfo**, **NWScanNSEntryInfo**

NWIntScanExtendedInfo

Scans a directory for the extended file information
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWIntScanExtendedInfo (
   NWCONN_HANDLE             conn,
   NWDIR_HANDLE              dirHandle,
   nuint8                    attrs,
   pnuint32                  iterHandle,
   pnstr8                    searchPattern,
   NW_EXT_FILE_INFO N_FAR   *entryInfo,
   nuint16                  augmentFlag);
```

Pascal Syntax

```
#include <nwdentry.inc>

Function NWIntScanExtendedInfo
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   attrs : nuint8;
   iterHandle : pnuint32;
   searchPattern : pnstr8;
   Var entryInfo : NW_EXT_FILE_INFO;
   augmentFlag : nuint16
) : NWCCODE ;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle for the directory to be scanned.

attrs

(IN) Specifies the search attributes.

iterHandle

(IN/OUT) Points to the search sequence number (-1 initially).

searchPattern

(IN) Points to the pattern for which to search (no wildcards are allowed).

entryInfo

(OUT) Points to the NW_EXT_FILE_INFO structure containing the extended file information.

augmentFlag

(IN) Specifies if wildcards are augmented:

0 = wildcards are not augmented
nonzero = wildcards are augmented

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8989	NO_SEARCH_PRIVILEGES
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89FF	NO_FILES_FOUND_ERROR

Remarks

NWIntScanExtendedInfo works only on files, not on directories.

All scanning is complete when the server returns 0x89FF.

NWIntScanExtendedInfo is synonymous with the **NWIntScanDirEntryInfo** function and uses an extension of the information structure.

The *iterHandle* parameter should point to 0xFFFFFFFF for the first call.

The *iterHandle* parameter should point to 0xFFFFFFFF for the first call.

The *attrs* parameter is used to include system and/or hidden files. If only the system bit is set in the *attrs* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files designated either hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The *attrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x10	$10	FA_DIRECTORY

The extended file information contains the information returned by the **NWIntScanDirEntryInfo** function plus the sizes of the data and resource forks. **NWIntScanExtendedInfo** also returns the physical size of a file.

NOTE: In the case of sparse files, the logical size may be much larger than the physical size.

NCP Calls

0x2222 22 40 Scan Directory Disk Space

See Also

NWIntScanDirEntryInfo, **NWScanNSEntryInfo**

NWIntScanFileInformation2

Scans the specified directory for the specified file (or directory) and returns the associated directory entry information in the DOS name space

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWIntScanFileInformation2 (
   NWCONN_HANDLE          conn,
   NWDIR_HANDLE           dirHandle,
   pnstr8                 filePattern,
   nuint8                 searchAttrs,
   pnuint8                iterHandle,
   NW_FILE_INFO2 N_FAR   *info,
   nuint16                augmentFlag);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWIntScanFileInformation2
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   filePattern : pnstr8;
   searchAttrs : nuint8;
   iterHandle : pnuint8;
   Var info : NW_FILE_INFO2;
   augmentFlag : nuint16;
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle relative to the *filePattern* parameter (or 0 if the *filePattern* parameter points to the complete path, including the volume name).

filePattern

(IN) Points to the string containing the file name or wildcard pattern to use in the search.

searchAttrs

(IN) Specifies the attributes to use for searching.

iterHandle

(IN/OUT) Inputs a pointer to the sequence number (set the first 4 bytes to 0xFF initially). Outputs a pointer to the 9-byte sequence number to be used for subsequent iterations.

info

(OUT) Points to the NW_FILE_INFO2 structure containing the file information.

augmentFlag

(IN) Specifies if wildcards are augmented:

0 = wildcards are not augmented

nonzero = wildcards are augmented

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8989	NO_SEARCH_PRIVILEGES
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89FF	NO_FILES_FOUND_ERROR

Remarks

The *searchAttrs* parameter includes system and/or hidden files. If only the system bit is set in the *searchAttrs* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files that are not

designated either hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The *searchAttrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x10	$10	FA_DIRECTORY

The *iterHandle* parameter points to a 9-byte identifier the server uses as an index for searching. In the first call to **NWIntScanFileInformation2**, the first 4 bytes of the number need to be set to 0xFF accomplished by typecasting the pointer to an nuint32, and assigning -1, or 0xFFFFFFFF to it. Every time **NWIntScanFileInformation2** is called, the sequence number for the next iteration is returned.

NCP Calls

0x2222 23 15 Scan File Information
0x2222 23 17 Get File Server Information
0x2222 87 02 Initialize Search
0x2222 87 03 Search For File Or Subdirectory

NWIntScanForTrustees

Scans a directory entry or file for trustees under the specified directory handle and path

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

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWIntScanForTrustees (
   NWCONN_HANDLE      conn,
   NWDIR_HANDLE       dirHandle,
   pnstr8             path,
   pnuint32           iterHandle,
   pnuint16           numOfEntries,
   NWET_INFO N_FAR   *entryTrusteeInfo,
   nuint16            augmentFlag);
```

Pascal Syntax

```
#include <nwdentry.inc>

Function NWIntScanForTrustees
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   iterHandle : pnuint32;
   numOfEntries : pnuint16;
   Var entryTrusteeInfo : NWET_INFO;
   augmentFlag : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle pointing to the directory or file to scan.

path

(IN) Points to an absolute directory or file path (if the *dirHandle*

parameter is not specified) or one relative to the *dirHandle* parameter (an absolute path must not be more than 255 bytes long).

iterHandle

(IN/OUT) Points to the server maintained sequence number (set to 0 initially).

numOfEntries

(OUT) Points to the buffer to receive the number of entries returned by **NWIntScanForTrustees**.

entryTrusteeInfo

(OUT) Points to the NWNET_INFO structure.

augmentFlag

(IN) Specifies if wildcards are augmented:

0 = wildcards are not augmented

nonzero = wildcards are augmented

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899C	NO_MORE_TRUSTEES

Remarks

For 3.x-4.x servers, **NWIntScanForTrustees** works for both files and directories. For 2.x servers, **NWIntScanForTrustees** works only for directories.

Directories can have any number of bindery objects as trustees. Trustees are returned in groups of 20 TRUSTEE_INFO structures. To obtain a complete list, set the *sequence* parameter to 0L for the initial call. **NWIntScanForTrustees**

do loop) until it returns 0x899C (NO_MORE_TRUSTEES). Because 0x899C also indicates INVALID_PATH, ensure the *dirHandle/path* parameter combination is correct.

Due to subtle differences in the operation of 2.x and 3.x servers, trustees may remain after an iteration, even though not all 20 positions are filled. If a position is not filled, the *objectID* parameter is set to 0L. Check the *objectID* parameter before printing each value in the *objectRights* parameter.

Both the *dirHandle* and *path* parameters must be in the default name space.

The default name space is the name space that matches the OS and the loaded name spaces on that volume. For example, Windows95 on a volume with OS/2 (LONG) name space will set OS/2 (LONG) name space as the default name space.

The *dirHandle* parameter can be zero if the *path* parameter points to the complete path, including the volume name. The *path* parameter can point to wildcard characters. However, only the first matching directory is scanned (as typical of 2.x servers).

NOTE: Call the **NWAllocTemporaryDirectoryHandle** function with the *path* parameter to check for a valid path.

The NWET_INFO structure receives trustee information. However, only the TRUSTEE_INFO structure is valid for servers 3.x and later. The first three fields, *entryName*, *creationDateAndTime*, and *ownerID* are valid only for 2.x servers. The *sequenceNumber* field should always be ignored.

NCP Calls

0x2222 22 12 Scan Directory For Trustees
0x2222 22 38 Scan File Or Directory For Extended Trustees
0x2222 23 17 Get File Server Information
0x2222 87 05 Scan File Or Subdirectory For Trustees

NWModifyMaximumRightsMask

Modifies the maximum rights mask of a directory
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWModifyMaximumRightsMask (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint8          revokeRightsMask,
   nuint8          grantRightsMask);
```

Pascal Syntax

```
#include<nwdirect.inc>

Function NWModifyMaximumRightsMask
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   revokeRightsMask : nuint8;
   grantRightsMask : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle for the directory whose maximum rights mask is being modified (or 0 if the *path* parameter points to the complete path, including the volume name).

path

(IN) Points to the absolute directory path (or a path relative to the directory handle) of the directory whose maximum rights mask is being modified.

revokeRightsMask

(IN) Specifies the rights being revoked.

grantRightsMask

(IN) Specifies the rights being granted.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	N0_MODIFY_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	Failure

Remarks

To modify the maximum rights mask for a directory, the requesting workstation must have parental rights to the directory.

The maximum rights mask follows:

Hex	Bit Definition
0x01	TA_READ
0x02	TA_WRITE
0x08	TA_CREATE
0x10	TA_DELETE

0x20	TA_OWNERSHIP
0x40	TA_SEARCH
0x80	TA_MODIFY

The rights specified by the *revokeRightsMask* parameter are deleted from the maximum rights mask for the directory, and the rights specified by the *grantRightsMask* parameter are added.

The maximum rights mask can be completely reset by setting the *revokeRightsMask* parameter to 0xFF and then setting the *grantRightsMask* parameter to the desired maximum rights mask. Maximum rights affect the specified directory only and are not inherited by subdirectories.

NCP Calls

0x2222 22 04 Modify Maximum Rights Mask
0x2222 23 17 Get File Server Information
0x2222 87 07 Modify File or SubDirectory DOS Information

See Also

NWGetEffectiveRights

NWMoveDirEntry (obsolete 6/96)

Moves or renames a directory entry (file or directory) on the same volume of a server but is now obsolete. Call the **NWIntMoveDirEntry** function instead.

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

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWMoveDirEntry
   (NWCONN_HANDLE   conn,
    nuint8          searchAttr,
    NWDIR_HANDLE    srcDirHandle,
    pnstr8          srcPath,
    NWDIR_HANDLE    dstDirHandle,
    pnstr8          dstPath);
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

searchAttr

(IN) Specifies the attributes to use in searching for the source entries.

srcDirHandle

(IN) Specifies the directory handle for the source directory (not optional, cannot be 0).

srcPath

(IN) Points to the source path (can include wildcards).

dstDirHandle

(IN) Specifies the NetWare directory handle for the destination directory.

dstPath

(IN) Points to the path name to use for the destination entry.

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

For NetWare 2.2, **NWMoveDirEntry (obsolete 6/96)** works only on files. The source directory (where the file resides) and the target directory (where the renamed file will be placed) do not need to be on the same volume.

To call **NWMoveDirEntry (obsolete 6/96)**, the client must have file modification privileges in both the source and the target directories.

The specified paths are relative to the specified directory handles. NetWare 3.11 and above accept paths relative to the directory handle, as well as full paths that include the volume.

The *searchAttr* parameter specifies the kind of entry to look for (hidden, system, etc.). If only the system bit is set in the *searchAttr* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files that are not designated either hidden or system are affected.

The *searchAttr* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The advantage of calling **NWMoveDirEntry (obsolete 6/96)** over DOS, OS/2, or other functions is its speed and efficiency. Since the move is within the server, the entry in the file system is simply deleted from the source and inserted in the destination. Directory entries move only on the file system level. There is no physical transfer of data between the source and the destination.

NCP Calls

0x2222 69 Rename File
0x2222 87 04 Rename Or Move A File Or Subdirectory

NWParseConfig

Parses a NET.CFG file
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: OS/2
Service: File System

Syntax

```
#include <nwconfig.h>
#include <nwerrors.h>
or
#include <nwcalls.h>

int N_API NWParseConfig (
   PCHAR                       configFile,
   PCHAR                       sectionName,
   UINT                        sectionInstance,
   UINT                        grammarTableSize,
   GrammarTableStruct N_FAR  *grammarTable,
   SetTableStruct N_FAR     *setTable);
```

Parameters

configFile

(IN) Points to an array containing the name of the NET.CFG file.

sectionName

(IN) Points to an array containing the name of the NET.CFG section. For example, for theLink Support Layer™ (LSL™) section, it would be LINK SUPPORT.

sectionInstance

(IN) Specifies the occurrence number of the section (generally 0 for the first one).

grammarTableSize

(IN) Specifies the number of GrammarTableStruct structures you are passing into **NWParseConfig** in the *grammarTable* parameter.

grammarTable

(OUT) Points to theGrammarTableStruct structure containing the grammar for which you are parsing.

setTable

(OUT) Points to an array ofSetTableStruct structures.

Return Values

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

0x0000	SUCCESSFUL

Remarks

The following constants are defined as follows:

```
8 MAX_PARAMETERS
 32 MAX_SECTION_NAME_SIZE
 80 MAX_VALUE_SIZE
 20 MAX_SET_ELEMENTS
```

Placing any of the types of values and optional types, except T_OPTIONAL, in the *paramType* field of the TypeDefaultStruct structure will cause **NWParseConfig** to try matching that value after the keyword.

For example, in the NETWARE Requester section of NET.CFG, one keyword is PREFERRED SERVER which is of type T_STRING. The parser will look for this keyword and then interpret the rest of the line as a string. If the type is T_OPTIONAL, you should place a default value in the *defaultValue* field of the TypeDefaultStruct structure to ensure a default value will be returned in the case of a missing value.

If the type is one of T_SET##, the *setTable* parameter should contain valid information to assist the parser, otherwise it should be NULL. A set is a list of values the value may contain.

The types of values and default values of optional types follow:

```
80h T_OPTIONAL
01h T_NUMBER
02h T_INDEX
03h T_STRING
04h T_HEX_STRING
05h T_HEX_NUMBER
06h T_LONG_NUMBER
07h T_LONG_HEX

equ 10h T_SET_1
equ 11h T_SET_2
equ 12h T_SET_3
equ 13h T_SET_4
equ 14h T_SET_5
equ 15h T_SET_6
equ 16h T_SET_7
equ 17h T_SET_8
equ 18h T_SET_9
equ 19h T_SET_10
```

```
equ 1Ah T_SET_11
equ 1Bh T_SET_12
equ 1Ch T_SET_13
equ 1Dh T_SET_14
equ 1Eh T_SET_15
equ 1Fh T_SET_16
```

NCP Calls

None

NWRenameDirectory

Renames a NetWare directory
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWRenameDirectory (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          oldName,
   pnstr8          newName);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWRenameDirectory
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   oldName : pnstr8;
   newName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle for the directory being deleted (or 0 if the *oldName* parameter points to the complete path, including the volume name).

oldName

(IN) Points to the string containing the name of the directory to be renamed.

newName

(IN) Points to the string containing the new directory name.

Return Values

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

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x88 36	NWE_PARAM_INVALID
0x89 80	FILE_IN_USE_ERROR
0x89 8B	NO_RENAME_PRIVILEGES
0x89 92	NO_FILES_RENAMED_NAME_EXISTS
0x89 96	SERVER_OUT_OF_MEMORY
0x89 98	VOLUME_DOES_NOT_EXIST
0x89 9B	BAD_DIRECTORY_HANDLE
0x89 9C	INVALID_PATH
0x89 9E	INVALID_FILENAME
0x89 A1	DIRECTORY_IO_ERROR
0x89 FD	BAD_STATION_NUMBER
0x89 FF	Failure

Remarks

The *newName* parameter should only include the new name of the directory without listing the volume or directory path. Otherwise, **NWRenameDirectory** will return NWE_PARAM_INVALID.

NCP Calls

0x2222 22 15 Rename Directory
0x2222 23 17 Get File Server Information

0x2222 87 04 Rename Or Move A File Or Subdirectory
0x2222 87 22 Generate Directory Base and Volume Number

See Also

NWCreateDirectory, **NWDeleteDirectory**

NWRenameFile

Allows a client to rename a file
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWRenameFile (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    oldDirHandle,
   pnstr8          oldFileName,
   nuint8          searchAttrs,
   NWDIR_HANDLE    newDirHandle,
   pnstr8          newFileName);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWRenameFile
  (conn : NWCONN_HANDLE;
   oldDirHandle : NWDIR_HANDLE;
   oldFileName : pnstr8;
   searchAttrs : nuint8;
   newDirHandle : NWDIR_HANDLE;
   newFileName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle containing the file.

oldDirHandle

(IN) Specifies the directory handle containing the file (or 0 if the *oldFileName* parameter points to the complete path, including the volume name).

oldFileName

(IN) Points to a string containing the original name of the file being renamed.

searchAttrs

(IN) Specifies the attributes to use in searching for the specified file.

newDirHandle

(IN) Specifies the new directory handle to contain the specified file.

newFileName

(IN) Points to a string containing the new name of the file.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8987	WILD_CARDS_IN_CREATE_FILE_NAME or CREATE_FILENAME_ERROR
0x898B	NO_RENAME_PRIVILEGES
0x898D	SOME_FILES_AFFECTED_IN_USE
0x898E	NO_FILES_AFFECTED_IN_USE
0x898F	SOME_FILES_AFFECTED_READ_ONLY
0x8990	NO_FILES_AFFECTED_READ_ONLY
0x8991	SOME_FILES_RENAMED_NAME_EXISTS
0x8992	NO_FILES_RENAMED_NAME_EXISTS
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899A	RENAMING_ACROSS_VOLUMES
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

0x89 A1	DIRECTORY_IO_ERROR
0x89 FD	BAD_STATION_NUMBER
0x89 FF	NO_FILES_FOUND_ERROR

Remarks

The source directory (where the file resides) and the target directory (where the renamed file is to be deposited) do not need to be the same directory. However, the two files must reside on the same server. **NWRenameFile** cannot move a file from one server to another or from one volume to another.

The *searchAttrs* parameter is used to include system and/or hidden files. If only the system bit is set in the *searchAttrs* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files that are not designated either hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The *searchAttrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

Since the path length is restricted to 256 bytes, applications must call the **NWAllocTemporaryDirectoryHandle** function to allocate the *dirHandle* parameter for path lengths greater than 256 bytes.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 69 Rename File
0x2222 87 04 Rename Or Move A File Or Subdirectory

See Also

NWAllocTemporaryDirectoryHandle, **NWPurgeErasedFiles**

NWRestoreDirectoryHandle

Restores a directory handle from its saved state
NetWare Server: 2.2
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWRestoreDirectoryHandle (
   NWCONN_HANDLE         conn,
   pnstr8                saveBuffer,
   NWDIR_HANDLE N_FAR   *newDirHandle,
   pnuint8               rightsMask);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWRestoreDirectoryHandle
  (conn : NWCONN_HANDLE;
   saveBuffer : pnstr8;
   Var newDirHandle : NWDIR_HANDLE;
   rightsMask : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

saveBuffer

(IN) Points to a 14-byte buffer in which the**NWSaveDirectoryHandle** function saved the directory handle information.

newDirHandle

(OUT) Points to the directory handle to be restored.

rightsMask

(OUT) Points to the rights mask for the directory to which the restored handle points.

Return Values

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

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 9B	BAD_DIRECTORY_HANDLE

NCP Calls

0x2222 22 24 Restore An Extracted Base Handle

NWSaveDirectoryHandle

Saves the information necessary to later restore a directory handle
NetWare Server: 2.2
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSaveDirectoryHandle (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          saveBuffer);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWSaveDirectoryHandle
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   saveBuffer : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle that is to be saved.

saveBuffer

(OUT) Points to a 14-byte buffer in which the directory handle information is to be saved.

Return Values

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

0x00	SUCCESSFUL

00	
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

NCP Calls

0x2222 22 23 Extract A Base Handle

NWScanConnectionsUsingFile

Scans all connections using a specified file
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanConnectionsUsingFile (
   NWCONN_HANDLE              conn,
   NWDIR_HANDLE               dirHandle,
   pnstr8                     filePath,
   pnint16                    iterHandle,
   CONN_USING_FILE N_FAR     *fileUse,
   CONNS_USING_FILE N_FAR   *fileUsed);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWScanConnectionsUsingFile
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   filePath : pnstr8;
   iterhandle : pnint16;
   Var fileUse : CONN_USING_FILE;
   Var fileUsed : CONNS_USING_FILE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired directory path.

filePath

(IN) Points to a full file path (or a path relative to *dirHandle*) specifying the file to be checked (wildcards are not allowed).

iterHnd

(IN/OUT) Points to the next record to be scanned (0 initially).

fileUse

(OUT) Points to the CONN_USING_FILE structure.

fileUsed

(OUT) Points to the CONNS_USING_FILE structure.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88FF	NWE_REQUESTER_FAILURE: Scan Completed
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A8	ERR_ACCESS_DENIED
0x89C6	NO_CONSOLE_PRIVILEGES

Remarks

You must have console operator rights to call **NWScanConnectionsUsingFile**.

Upon each subsequent call, the number of the next record to be scanned is returned in the *iterHnd* parameter. This value should not be changed during the scan. **NWScanConnectionsUsingFile** returns 0xFFFFFFFF upon completion.

If no connections are using the specified file, the structure returned by the *fileUsed* parameter will contain zeroes. Check the *connCount* parameter in the returned structure to see the number of connections actually using the file.

If the *fileUse* parameter is NULL, the records are returned in the *fileUsed* parameter in groups, instead of one at a time.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 220 Get Connections Using A File (2.x)
0x2222 23 236 Get Connections Using A File (3.x-4.x)
0x2222 23 244 Convert Path To Dir Entry

NWScanDirectoryForTrustees2

Scans a directory for trustees using the specified path and directory handle
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanDirectoryForTrustees2 (
   NWCONN_HANDLE          conn,
   NWDIR_HANDLE           dirHandle,
   pnstr8                 srchPath,
   pnuint32               iterHandle,
   pnstr8                 dirName,
   pnuint32               dirDateTime,
   pnuint32               ownerID,
   TRUSTEE_INFO N_FAR  *trusteeList);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWScanDirectoryForTrustees2
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   searchPath : pnstr8;
   iterHandle : pnuint32;
   dirName : pnstr8;
   dirDateTime : pnuint32;
   ownerID : pnuint32;
   Var trusteeList : TRUSTEE_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle for the directory being scanned (0 if the *srchPath* parameter points to the complete path, including the volume name).

srchPath

srchPath

(IN) Points to an absolute directory path (or a path relative to the directory handle) and a search pattern.

iterHandle

(IN/OUT) Points to the sequence number to be used for subsequent calls (0 initially).

dirName

(OUT) Points to the directory name found (optional, up to 256 bytes).

dirDateTime

(OUT) Points to the creation date and time of the directory (optional).

ownerID

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

trusteeList

(OUT) Points to an array of 20 TRUSTEE_INFO structures.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	N0_MODIFY_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	NO_MORE_TRUSTEES

Remarks

The *srchPath* parameter can include wildcard characters.

Directories can have any number of Bindery objects as trustees. The directory trustees are stored and retrieved in groups on the server. To

obtain a complete list, use the *iterHandle* parameter.

NWScanDirectoryForTrustees2 increments the value referenced by the *iterHandle* parameter to the next appropriate value. For subsequent calls, pass in the new value of the *iterHandle* parameter.

Trustees are returned in groups of 20 TRUSTEE_INFO structures. Due to subtle differences in the operation of 2.x and 3.x servers, trustees may remain after an iteration, even though not all 20 positions are filled. If a position is not filled, the *ownerID* parameter points to a value of 0L.

NWScanDirectoryForTrustees2 should be called until it returns 0x899C (NO_MORE_TRUSTEES). Because 0x899C also means INVALID_PATH, ensure the *dirHandle/pbstrSrchPath* parameter combination is correct.

NULL can be substituted for all optional items. However, all parameter positions must be filled.

NCP Calls

0x2222 22 1 Get Directory Path
0x2222 22 2 Scan Directory Information
0x2222 22 12 Scan Directory For Trustees
0x2222 22 38 Trustees Scan Ext
0x2222 23 17 Get File Server Information
0x2222 87 02 Initialize Search
0x2222 87 03 Search For File or Subdirectory
0x2222 87 05 Scan File Or Subdirectory For Trustees
0x2222 87 06 Obtain File or Subdirectory Information

NWScanDirectoryInformation2 (obsolete 6/96)

Returns directory information for a directory specified by the connection handle, directory handle, and directory path but is now obsolete. Call the **NWIntScanDirectoryInformation2** function instead.

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

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanDirectoryInformation2
   (NWCONN_HANDLE    conn,
    NWDIR_HANDLE     dirHandle,
    pnstr8           srchPath,
    pnuint8          sequence,
    pnstr8           dirName,
    pnuint32         dirDateTime,
    pnuint32         ownerID,
    pnuint8          rightsMask);
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle for the directory being scanned (0 if the *srchPath* parameter points to the complete path, including the volume name).

srchPath

(IN) Points to an absolute directory path (or a path relative to the directory handle) and a search pattern (optional, wildcards can be used).

sequence

(IN/OUT) Points to a 9-byte sequence number to be used for subsequent calls (the first 4 bytes should be 0xFF initially).

dirName

(OUT) Points to the directory name found (256 bytes, optional).

dirDateTime

(OUT) Points to the creation date and time of the directory (4 bytes,

optional).

ownerID

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

rightsMask

(OUT) Points to the maximum rights mask for the directory found (optional).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89FF	NO_FILES_FOUND_ERROR

Remarks

Under NETX, if an invalid connection handle is passed to the *conn* parameter, **NWScanDirectoryInformation2 (obsolete 6/96)** will return 0x0000. NETX will pick a default connection handle if the connection handle cannot be resolved.

All parameter fields must be filled. However, NULL may be substituted in parameters where no information is desired.

If wildcards are used in the *srchPath* parameter, only the directory information for the first matching directory is returned.

The *dirName* parameter is ignored when returning volume information for NetWare 2.2 servers.

The *dirDateTime* parameter does not point to valid information on 2.x servers and does not point to valid information for the root directory (volume) on 2.2 servers. Set the *dirDateTime* parameter to zero when

getting volume information for 2.2 servers. For volumes, 2.x servers do not return a directory name, and the *dirDateTime* parameter should always point to zero.

The *rightsMask* parameter points to the maximum rights mask of the subdirectory. The bits in the maximum rights mask are defined as follows:

```
0x00 = TA_NONE
0x01 = TA_READ
0x02 = TA_WRITE
0x04 = TA_OPEN
0x08 = TA_CREATE
0x10 = TA_DELETE
0x20 = TA_OWNERSHIP
0x40 = TA_SEARCH
0x80 = TA_MODIFY
0xFB = TA_ALL
```

NOTE: TA_OPEN is obsolete in 3.x and above.

NCP Calls

0x2222 22 1 Get Directory Path
0x2222 22 02 Scan Directory Information
0x2222 23 17 Get File Server Information
0x2222 87 02 Initialize Search
0x2222 87 03 Search For File Or Subdirectory
0x2222 87 06 Obtain File Or Subdirectory Information

See Also

NWParseNetWarePath

NWScanDirEntryInfo (obsolete 6/96)

Obtains information about 3.x and 4.x directory entries (files or directories) but is now obsolete. Call **NWIntScanDirEntryInfo** instead.

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanDirEntryInfo
  (NWCONN_HANDLE          conn,
   NWDIR_HANDLE           dirHandle,
   nuint16                attrs,
   pnuint32               iterHandle,
   pnuint8                searchPattern,
   NWENTRY_INFO N_FAR      *entryInfo);
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle indexing the directory to scan (not optional, cannot be 0).

attrs

(IN) Specifies the attributes to be used for the scan.

iterHandle

(IN/OUT) Points to an nuint32 buffer to receive the search sequence from the server (-1 initially).

searchPattern

(IN) Points to the name of the entry for which to scan (wildcards are allowed).

entryInfo

(OUT) Points to the NWENTRY_INFO structure.

Return Values

These are common return values; see Return Values for more

information.

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 89	NO_SEARCH_PRIVILEGES
0x89 98	VOLUME_DOES_NOT_EXIST
0x89 9B	BAD_DIRECTORY_HANDLE
0x89 9C	INVALID_PATH
0x89 FF	Failure

Remarks

NWScanDirEntryInfo (obsolete 6/96) will not support NetWare augmented wildcard characters unless you have manually set the high-order bits. For example, using non-augmented wildcards, *.* will match any name with a period, while * will match any string. See the explanation under "wildcard characters" in the Glossary.

The server manages the information after **NWScanDirEntryInfo (obsolete 6/96)** is called the first time. All scanning is complete when the server returns 0x89FF.

The *searchPattern* parameter cannot point to any path elements and the *dirHandle* parameter must index the complete path.

On the first call, the *iterHandle* parameter should point to 0xFFFFFFFF. After that, the server manages the information. All scanning is complete when the server returns 0x89FF.

NWScanDirEntryInfo (obsolete 6/96) can also be used to scan for information about other directories, including the root directory. In this mode, the *dirHandle* parameter needs to index the root or a directory. The *searchPattern* parameter needs to point to a NULL value.

NWScanDirEntryInfo (obsolete 6/96) works with DOS name spaces only. To scan using alternate name spaces, convert the path to DOS name space by calling either the **NWGetNSPath** or **NWScanNSEntryInfo** function.

NWScanDirEntryInfo (obsolete 6/96) works for servers 3.11 and above only.

The *attrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x0000	$0000	SA_NORMAL
0x0002	$0002	SA_HIDDEN
0x0004	$0004	SA_SYSTEM
0x8000	$8006	SA_ALL

The NWENTRY_INFO structure should be initialized to 0 before **NWScanDirEntryInfo (obsolete 6/96)** is called the first time.

NCP Calls

0x2222 22 01 Get Directory Path
0x2222 22 30 Scan A Directory
0x2222 22 31 Get Directory Entry

See Also

NWGetNSInfo, NWIntScanExtendedInfo, NWScanNSEntryInfo

NWScanExtendedInfo (obsolete 6/96)

Scans a directory for the extended file information but is now obsolete. Call **NWIntScanExtendedInfo** instead.

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanExtendedInfo
  (NWCONN_HANDLE             conn,
   NWDIR_HANDLE              dirHandle,
   nuint8                    attrs,
   pnuint32                  iterHandle,
   pnstr8                    searchPattern,
   NW_EXT_FILE_INFO N_FAR     *entryInfo);
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle for the directory to be scanned.

attrs

(IN) Specifies the search attributes.

iterHandle

(IN) Points to the search sequence number (-1 initially).

searchPattern

(IN) Points to the pattern for which to search (no wildcards are allowed).

entryInfo

(OUT) Points to theNW_EXT_FILE_INFO structure containing the extended file information.

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

NWScanExtendedInfo (obsolete 6/96) works only on files, not on directories.

The *iterHandle* parameter should point to 0xFFFFFFFF for the first call.

The *attrs* parameter is used to include system and/or hidden files. If only the system bit is set in the *attrs* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files designated either hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The *attrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

NOTE: In the case of sparse files, the logical size may be much larger than the physical size.

NCP Calls

0x2222 22 40 Scan Directory Disk Space

See Also

NWScanNSEntryInfo

NWScanFileInformation2 (obsolete 6/96)

Scans the specified directory for the specified file (or directory) and returns the associated directory entry information but is now obsolete. Call **NWIntScanFileInformation2** instead.

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

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanFileInformation2
  (NWCONN_HANDLE          conn,
   NWDIR_HANDLE           dirHandle
   pnstr8                 filePattern,
   nuint8                 searchAttrs,
   pnuint8                iterHandle,
   NW_FILE_INFO2 N_FAR     *info);
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle relative to the *filePattern* parameter (0 if the *filePattern* parameter points to the complete path, including the volume name).

filePattern

(IN) Points to the string containing the file name or wildcard pattern to use in the search.

searchAttrs

(IN) Specifies the attributes to use for searching.

iterHandle

(IN/OUT) Inputs a pointer to the sequence number (the first 4 bytes should be 0xff initially). Outputs a pointer to the 9-byte sequence number to be used for subsequent iteration.

info

(OUT) Points to the NW_FILE_INFO2 structure containing the file information.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

The *searchAttrs* parameter includes system and/or hidden files. If only the system bit is set in the *searchAttrs* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files that are not designated either hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The *searchAttrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

Typecast the *iterHandle* parameter to nuint32, and assign -1, or 0xFFFFFFFF to it. Every time **NWScanFileInformation2 (obsolete 6/96)** is called, the sequence number for the next iteration is returned.

NCP Calls

0x2222 23 15 Scan File Information

NWScanOpenFilesByConn2

Scans information about the files opened by a specified connection
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanOpenFilesByConn2 (
   NWCONN_HANDLE               conn,
   NWCONN_NUM                  connNum,
   pnint16                     iterHandle,
   OPEN_FILE_CONN_CTRL N_FAR  *openCtrl,
   OPEN_FILE_CONN N_FAR       *openFile);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWScanOpenFilesByConn2
  (conn : NWCONN_HANDLE;
   connNum : NWCONN_NUM;
   iterHandle : pnint16;
   Var openCtrl : OPEN_FILE_CONN_CTRL;
   Var openFile : OPEN_FILE_CONN
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

connNum

(IN) Specifies the connection number of the logged-in object to be scanned.

iterHandle

(IN/OUT) Points to the next record to be scanned (0 initially).

openCtrl

(OUT) Points to theOPEN_FILE_CONN_CTRL structure.

openFile

(OUT) Points to theOPEN_FILE_CONN structure.

(OUT) Points to the OPEN_FILE_CONN structure.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88FF	Scan Completed
0x89FD	BAD_STATION_NUMBER

Remarks

For 2.x and 3.x, you must have console operator rights to call **NWScanOpenFilesByConn2** or NO_CONSOLE_PRIVILEGES will be returned.

For 4.x, you can call **NWScanOpenFilesByConn2** to return information about the connection without needing console operator privileges. To return information about other connection numbers, you must have console rights. A client with console privileges can pass any valid connection number to **NWScanOpenFilesByConn2** and receive information about that connection.

Upon each subsequent call, the *iterHandle* parameter returns the number of the next record to be scanned and points to 0xFFFFFFFF upon completion. It should not be changed during the scan.

The OPEN_FILE_CONN_CTRL structure is used internally and should not be written to.

The *parent* and *forkCount* fields in the OPEN_FILE_CONN structure are not set by 2.x and are filled with 0xFF. In 2.x, the *nameSpace* field is always set to 0.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 219 Get Connection's Open Files (2.x)
0x2222 23 235 Get Connection's Open Files (3.x-4.x)

See Also

NWGetPathFromDirectoryEntry, NWGetPathFromDirectoryBase

NWSetCompressedFileLengths

Sets the uncompressed and compressed lengths of a file
Local Servers: blocking
Remote Servers: blocking
Classification: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwfile.h>

int NWSetCompressedFileLengths (
   int    handle,
   LONG   uncompressedLength,
   LONG   compressedLength;
```

Parameters

handle

(IN) Specifies the handle of the file for which to set the lengths.

uncompressedLength

(IN) Specifies the length of the file in an uncompressed state.

compressedLength

(IN) Specifies the length of the file after being compressed.

Return Values

0x00	Success
0xFF	Failure

Remarks

NWSetCompressedFileLengths sets the compressed and uncompressed lengths of a file.

NWSetCompressedFileLengths is useful for restoring directory entry information about files that have previously been backed up.

The *uncompressedLength* parameter is the length normally seen in normal directory listings.

See Also

NWGetCompressedFileLengths

NWSetCompressedFileSize

Attempts to set the logical file size for a compressed file
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetCompressedFileSize (
   NWCONN_HANDLE   conn,
   nuint32         fileHandle,
   nuint32         reqFileSize,
   pnuint32        resFileSize);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWSetCompressedFileSize
  (conn : NWCONN_HANDLE;
   fileHandle : nuint32;
   reqFileSize : nuint32;
   resFileSize : pnuint32
) : NWCCODE;
```

Parameters

conn

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

fileHandle

(IN) Specifies an OS or NetWare file handle.

reqFileSize

(IN) Specifies the requested file size.

resFileSize

(OUT) Points to the size actually assigned by the OS.

Return Values

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

0x00 00	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 88	INVALID_FILE_HANDLE
0x89 A8	ERR_ACCESS_DENIED

Remarks

The logical file size is the true size of the file as reported by the client operating systems. When a file is compressed, it shrinks in physical size. However, its logical size should remain the same. In cases where the client forces the creation of a compressed file (by opening a file in compressed mode), the NetWare OS gets the actual size of the file by calling **NWSetCompressedFileSize**.

If the *fileHandle* parameter contains a NetWare handle, the *conn* parameter contains the connection handle of the associated server. If NETX is running and a DOS file handle is passed, the *conn* parameter must also contain a valid connection ID. In all other circumstances, the *conn* parameter is ignored.

NCP Calls

0x2222 90 12 Set Compressed File Size

NWSetDirectoryHandlePath

Sets the target directory handle for the specified directory handle and path
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetDirectoryHandlePath (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    sourceDirHandle,
   pnstr8          dirPath,
   NWDIR_HANDLE    destDirHandle);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWSetDirectoryHandlePath
  (conn : NWCONN_HANDLE;
   sourceDirHandle : NWDIR_HANDLE;
   dirPath : pnstr8;
   destDirHandle : NWDIR_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

sourceDirHandle

(IN) Specifies the source directory handle (index number) identifying the volume or directory on a NetWare server being reassigned (1-255).

dirPath

(IN) Points to the source directory path (optional).

destDirHandle

(IN) Specifies the target directory handle (index number) to become the new directory handle for the specified directory.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FA	TEMP_REMAP_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	Failure

Remarks

If **NWSetDirectoryHandlePath** fails, the *destDirHandle* parameter remains unchanged.

In cases where multiple NetWare servers are being used, the *sourceDirHandle* and *destDirHandle* paramteters must have the same server connection handle identifier.

NWSetDirectoryHandlePath assigns the *destDirHandle* parameter to a directory path defined by combining the *sourceDirHandle* parameter and the string accessed by the *dirPath* parameter.

A NetWare server maintains a Directory Handle Table for each workstation that is logged in.

The *destDirHandle* parameter is another index number from the Directory Handle Table for the NetWare server.

The *dirPath* parameter can identify a full or partial directory path. A full directory path defines a volume or a directory on a given NetWare server in the format VOLUME:DIRECTORY/.../DIRECTORY. A partial

directory path specifies at least a directory and one or more parent directories.

Applications frequently combine a directory handle and a directory path to specify a target directory. For example, if the specified directory handle points to SYS: and the specified directory path is PUBLIC/WORDP, the specified directory is SYS:PUBLIC/WORDP.

When an application defines a target directory using only a directory handle, the application must set the *dirPath* parameter to a NULL string. When an application defines a directory using only a directory path, the application must set the *sourceDirHandle* parameter to zero.

NCP Calls

0x2222 22 00 Set Directory Handle
0x2222 23 17 Get File Server Information
0x2222 87 09 Set Short Directory Handle

See Also

NWGetDirectoryHandlePath

NWSetDirectoryInformation

Changes information about a directory including the creation date and time, owner object ID, and maximum rights mask

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

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetDirectoryInformation (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint32         dirDateTime,
   nuint32         ownerID,
   nuint8          rightsMask);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWSetDirectoryInformation
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   dirDateTime : nuint32;
   ownerID : nuint32;
   rightsMask : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle (index number from 1-255) pointing to the directory, partial directory, or volume whose information is being set (0 if the *path* parameter points to the complete path, including the volume name).

path

(IN) Points to the directory path of the directory being changed.

dirDateTime

(IN) Specifies the new creation date and time.

ownerID

(IN) Specifies the bindery object ID of the owner who created the directory.

rightsMask

(IN) Specifies the new maximum rights mask for the directory.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	N0_MODIFY_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89F0	WILD_CARD_NOT_ALLOWED
0x89FF	Failure, NO_FILES_FOUND_ERROR

Remarks

NWSetDirectoryInformation defines the target directory by passing a directory handle and a directory path.

NOTE: Volume information for 2.x servers cannot be modified. The *dirDateTime* and *ownerID* parameters cannot be set for volumes on 2.x servers.

A NetWare server maintains a Directory Handle Table for each logged in workstation.

The *path* parameter cannot contain wild card characters or **NWSetDirectoryInformation** will return WILD_CARD_NOT_ALLOWED.

The *path* parameter can identify a full or partial directory path. A full directory path defines a volume or a directory on a given NetWare server in the format VOLUME:DIRECTORY/.../DIRECTORY. A partial directory path specifies at least a directory, and possibly one or more parent directories.

Applications frequently combine a directory handle and a directory path to specify a target directory. For example, if the specified directory handle points to SYS: and the specified directory path is PUBLIC/WORDP, the specified directory is SYS:PUBLIC/WORDP.

The *dirDateTime* parameter appears in standard DOS format. The first two bytes contain the year (7 bits), month (4 bits), and day (5 bits) fields, and the second two bytes contain the hour (5 bits), minute (6 bits), and second (5 bits) fields.

NWSetDirectoryInformation sets the date and time in ascending order (byte 1, byte 2, byte 3, byte 4). The date and time values are defined as follows:

Type	Value
Year	0=1980, 1=1981, ..., 119=2099
Month	1 to 12
Day	1 to 31
Hour	0 to 23
Minute	0 to 59
Second	0 to 29 (in units of 2 seconds)

The *rightsMask* parameter contains the maximum rights mask for the subdirectory. The bits in the maximum rights mask are defined as follows:

```
0x00 = TA_NONE
0x01 = TA_READ
0x02 = TA_WRITE
0x04 = TA_OPEN
0x08 = TA_CREATE
0x10 = TA_DELETE
```

```
0x20 = TA_OWNERSHIP
0x40 = TA_SEARCH
0x80 = TA_MODIFY
0xFB = TA_ALL
```

NOTE: TA_OPEN is obsolete in version 3.x and above.

To change information for a directory, the requesting workstation must have parental and modify rights to the directory's parent. Only a workstation with SUPERVISOR rights can change the owner of a directory.

NCP Calls

0x2222 22 25 Set Directory Information
0x2222 23 17 Get File Server Information
0x2222 87 07 Modify File Or Subdirectory DOS Information

See Also

NWParseNetWarePath

NWSetDirEntryInfo

Changes information about a directory entry (file or directory)
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetDirEntryInfo (
   NWCONN_HANDLE          conn,
   NWDIR_HANDLE           dirHandle,
   nuint8                 searchAttrs,
   nuint32                iterHandle,
   nuint32                changeBits,
   NWENTRY_INFO N_FAR  *newEntryInfo);
```

Pascal Syntax

```
#include <nwdentry.inc>

Function NWSetDirEntryInfo
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   searchAttrs : nuint8;
   iterHandle : nuint32;
   changeBits : nuint32;
   Var newEntryInfo : NWENTRY_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle.

searchAttrs

(IN) Specifies the search attribute to use in searching for the directory entry.

iterHandle

(IN) Is currently unused and ignored.

changeBits

(IN) Specifies the set of bits to indicate which attributes to change.

newEntryInfo

(IN) Points to the NWENTRY_INFO structure.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

NWSetDirEntryInfo only works with 3.11 and above servers.

For files, the *dirHandle* parameter must point to parent directory. For directories, it should follow the same conventions as for the **NWIntScanDirEntryInfo** function.

The *searchAttrs* parameter specifies the kind of entry to look for (hidden, system, etc.). For example, if only the system bit is set in the *searchAttrs* parameter, all files except hidden files are affected. If only the hidden bit is set, all files except system files are affected. If neither bit is set (0x00), only files not designated either hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The *searchAttrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL

0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x10	$10	FA_DIRECTORY

The bit definition for the *changeBits* parameter follows:

C Value	Pascal Value	Value Name
0x0001L	$0001	MModifyNameBit
0x0002L	$0002	MFileAttributesBit
0x0004L	$0004	MCreateDateBit
0x0008L	$0008	MCreateTimeBit
0x0010L	$0010	MOwnerIDBit
0x0020L	$0020	MLastArchivedDateBit
0x0040L	$0040	MLastArchivedTimeBit
0x0080L	$0080	MLastArchivedIDBit
0x0100L	$0100	MLastUpdatedDateBit
0x0200L	$0200	MLastUpdatedTimeBit
0x0400L	$0400	MLastUpdatedIDBit
0x0800L	$0800	MLastAccessedDateBit
0x1000L	$1000	MInheritedRightsMaskBit
0x2000L	$2000	MMaximumSpaceBit

The NWENTRY_INFO structure must be initialized to 0 before calling the **NWIntScanDirEntryInfo** function.

To change information for a directory, the requesting workstation must have access control and modify rights. Only a workstation with SUPERVISOR rights can change the owner of a directory. The *lastModifyDateAndTime* field in the NWDIR_INFO structure cannot be changed for volumes. Otherwise, the last modified date and time will be set to the current date and time.

For files, the *dirHandle* parameter must point to the parent directory. The *nameLength* and *name* fields in the NWENTRY_INFO structure must contain the specific file information.

For directories, if the *dirHandle* parameter points to the parent directory, the *nameLength* and *name* fields in the NWENTRY_INFO structure must contain the specific directory information.

For directories, if the *dirHandle* parameter points to the specific directory itself, the *nameLength* field must be set to 0.

For each name space, the *dirHandle* parameter and the *nameSpace*, *name* and *nameLength* fields must be synchronized to indicate the correct name space.

NCP Calls

0x2222 22 37 Set Directory Entry Information
0x2222 23 17 Get File Server Information
0x2222 87 07 Modify File Or Subdirectory DOS Information

See Also

NWIntScanDirEntryInfo, **NWSetNSEntryDOSInfo**

NWSetExtendedFileAttributes2

Sets the extended attributes of a file
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetExtendedFileAttributes2 (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint8          extAttrs);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWSetExtendedFileAttributes2
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   extAttrs : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the connection handle.

dirHandle

(IN) Specifies the directory handle of the root directory of the new directory.

path

(IN) Points to the string containing the name and path of the new directory.

extAttrs

(IN) Specifies the extended attributes for the file.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	N0_MODIFY_PRIVILEGES
0x898D	SOME_FILES_AFFECTED_IN_USE
0x898E	NO_FILES_AFFECTED_IN_USE
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	Failure, NO_FILES_FOUND_ERROR

Remarks

NWSetExtendedFileAttributes2 requires Search rights to the directory where the file resides.

The *path* parameter can specify either the complete path name for a file or a path relative to the current working directory.

For example, if the complete path name is SYS:ACCOUNT/DOMEST/TARGET.DAT and the directory handle mapping is SYS:ACCOUNT, the *path* parameter could point to either of the following:

```
SYS:/ACCOUNT/DOMEST/TARGET.DAT or DOMEST/TARGET.DAT
```

The bit map for the *extAttrs* parameter follows:

Bits	Function
0-2	Search mode bits
4	Transaction bit
5	Index bit (2.x only)
6	Read audit bit (not yet implemented)
7	Write audit bit (not yet implemented)

Setting the transaction bit prompts TTS™ to track all Writes to the file during a transaction. A transaction file cannot be deleted or renamed until the transaction bit is turned off by calling **NWSetExtendedFileAttributes2**.

Setting the index bit prompts NetWare to index the File Allocation Tables for the file, thereby reducing the time required to access files. Files larger than 2MB should have this bit set.

NCP Calls

0x2222 79 Set File Extended Attribute

See Also

NWGetExtendedFileAttributes2

NWSetDirSpaceLimit

Specifies a space limit (in 4 KB blocks) on a particular subdirectory
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetDirSpaceLimit (
   NWCONN_HANDLE    conn,
   NWDIR_HANDLE     dirHandle,
   nuint32          spaceLimit);
```

Pascal Syntax

```
#include <nwdirect.inc>

Function NWSetDirSpaceLimit
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   spaceLimit : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the NetWare directory handle pointing to the directory to scan.

spaceLimit

(IN) Specifies the number of 4K blocks needed to limit the directory space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8901	ERR_INSUFFICIENT_SPACE
0x898C	N0_MODIFY_PRIVILEGES
0x89BF	INVALID_NAME_SPACE

Remarks

If the space limit is set to 0, space limit restrictions are lifted. If the restriction is 0xFFFFFFFF, the space limit on the directory is set to 0.

NCP Calls

0x2222 22 36 Set Directory Disk Space Restrictions

NWSetFileAttributes

Modifies a file's original attributes
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetFileAttributes (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          fileName,
   nuint8          searchAttrs,
   nuint8          newAttrs);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWSetFileAttributes
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   fileName : pnstr8;
   searchAttrs : nuint8;
   newAttrs : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle containing the file.

dirHandle

(IN) Specifies the NetWare directory handle (0 if the *fileName* parameter points to the complete path, including the volume name).

fileName

(IN) Points to the string containing a path name, relative to *dirHandle*.

searchAttrs

(IN) Specifies the attributes to use in searching for a file.

newAttrs

(IN) Specifies the new attributes to be applied to the file designated by the *dirHandle* and *pbstrFileName* parameters.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	N0_MODIFY_PRIVILEGES
0x898D	SOME_FILES_AFFECTED_IN_USE
0x898E	NO_FILES_AFFECTED_IN_USE
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	Failure, NO_FILES_FOUND_ERROR

Remarks

The *fileName* parameter can specify either a complete path name or a path relative to the current working directory. For example, if the complete path name is SYS:ACCOUNT/DOMEST/TARGET.DAT and the directory handle mapping is SYS:ACCOUNT, the *fileName* parameter could point to either of the following:

```
SYS:ACCOUNT/DOMEST/TARGET.DAT or
DOMEST/TARGET.DAT
```

The *searchAttrs* parameter includes system and/or hidden files. If only the system bit is set in the *searchAttrs* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files that are not designated either hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The *searchAttrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 70 Set File Attributes
0x2222 87 07 Modify File Or Subdirectory DOS Information

See Also

NWGetExtendedFileAttributes2, **NWIntScanFileInformation2**, **NWSetFileInformation2**

NWSetFileInformation2

Updates file information
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: File System

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetFileInformation2
  (NWCONN_HANDLE          conn,
   NWDIR_HANDLE           dirHandle,
   pnstr8                 fileName,
   nuint8                 searchAttrs,
   NW_FILE_INFO2 N_FAR  *info);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWSetFileInformation2
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   fileName : pnstr8;
   searchAttrs : nuint8;
   Var info : NW_FILE_INFO2
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle containing the file to be modified.

dirHandle

(IN) Specifies the NetWare directory handle (0 if the *fileName* parameter points to the complete path, including the volume name).

fileName

(IN) Points to the name of the file to modify.

searchAttrs

(IN) Specifies the search attributes.

info

(IN) Points to the NW_FILE_INFO2 structure.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE
0x898C	N0_MODIFY_PRIVILEGES
0x898E	NO_FILES_AFFECTED_IN_USE
0x8994	NO_WRITE_PRIVILEGES_OR_READONLY
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89FC	NO_SUCH_OBJECT
0x89FD	BAD_STATION_NUMBER
0x89FE	DIRECTORY_LOCKED
0x89FF	Failure, NO_FILES_FOUND_ERROR

Remarks

NWSetFileInformation2 handles long names (up to 256 bytes).

NWSetFileInformation2 sets the file information defined by the NW_FILE_INFO2 structure.

The *fileName* parameter can specify either a complete path name or a path relative to the current working directory. For example, if the complete path name is SYS:ACCOUNT/DOMEST/TARGET.DAT, and the directory handle mapping is SYS:ACCOUNT, the *fileName* parameter could be either of the following:

```
SYS:ACCOUNT/DOMEST/TARGET.DAT or
DOMEST/TARGET.DAT
```

The *searchAttrs* parameter is used to include system and/or hidden files. If only the system bit is set in the *searchAttrs* parameter, all files are affected except hidden files. If only the hidden bit is set, all files are affected except system files. When neither bit is set (0x00), only files that are not designated hidden or system are affected.

NOTE: A file is designated hidden or system if its corresponding file attribute is set.

The *searchAttrs* parameter can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

NCP Calls

0x2222 23 16 Set File Information
0x2222 23 17 Get File Server Information
0x2222 87 07 Modify File Or Subdirectory DOS Information

See Also

NWGetExtendedFileAttributes2, **NWIntScanFileInformation2**, **NWSetFileAttributes**,

NWSetNetWareErrorMode

Sets the NetWare error handling mode for the requesting workstation
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, Windows 3.1
Service: File System

Syntax

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

NWCCODE N_API NWSetNetWareErrorMode (
   nuint8    errorMode,
   pnuint8   prevMode);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWSetNetWareErrorMode
  (errorMode : nuint8;
   prevMode : pnuint8
) : NWCCODE;
```

Parameters

errorMode

(IN) Specifies the error mode.

prevMode

(OUT) Points to the previous error mode (optional).

Return Values

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

0x0000	SUCCESSFUL

Remarks

NWSetNetWareErrorMode fails when VLM.EXE is running.

The *errorMode* parameter can contain of the following three values (the default value is 0):

0	User intervention is required for NetWare file I/O critical errors.
1	Specific error code is returned to the calling program from the NetWare Shell for all NetWare file I/O errors.
2	DOS error code is returned to the calling program from the NetWare Shell for all NetWare file I/O errors.

NCP Calls

None

NWVolumeIsCDROM

Determines whether a given volume is a CDROM volume
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
Platform: NLM
SMP Aware: Yes
Service: File System

Syntax

```
#include <nwdir.h>

int NWVolumeIsCDROM (
   LONG   volumeNumber,
   LONG  *isCDROM;
```

Parameters

volumeNumber

(IN) Specifies the number of the volume to be queried.

isCDROM

(OUT) Points to either TRUE or FALSE, indicating whether the volume is a CDROM volume.

Return Values

0	ESUCCESS
0xFF FF	Failure---NWErrno is set with the appropriate error code.

Remarks

NWIsVolumeCDROM allows the caller to determine whether a give volume is a CDROM volume.

The **NWSetCompressedFileLengths** function fails if CDROM.NLM is not loaded of if the queried volume is not mounted.

See Also

NWGetExtendedVolumeInfo

NWGetExtendedVolumeInfo, NWGetVolumeName

opendir

Opens a directory for reading with the **readdir** function
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <dirent.h>

DIR *opendir (
   const char  *pathname);
```

Parameters

pathname

(IN) Can be either relative to the current working directory or it can be an absolute path name.

Return Values

Returns a pointer to a structure (required for subsequent calls to the **readdir** function) containing the file names matching the pattern specified by the *pathname* parameter.

Returns NULL if the path name is not valid or if there are no files matching the path name. If an error occurs, errno and NetWareErrno are set.

Remarks

The last part of the path name can contain the characters `?' and `*' for matching multiple files.

More than one directory can be read at the same time using the **opendir**, **readdir**, and **closedir** functions.

opendir calls the **malloc** function to allocate memory for a DIR structure. The **closedir** function frees the memory.

NOTE: Information about the first file or directory matching the specified path name is not placed in the DIR structure until after the first call to the **readdir** function.

The **SetCurrentNameSpace** function sets the name space that is used for parsing the path input to **opendir**.

Beginning with Release 9 of NW SDK, **opendir** returns long names in the *d_name* field of the dirent structure if the target namespace is previously set (by calling the **SetTargetNameSpace** function) to something other than DOS. You must compile with the dirent.h file included with Release 9 or later and link with the new nwpre.obj and is valid only when calling **opendir** to the local server.

NOTE: The position in the old structure of the*d_name* field has been assumed by the new *d_nameDOS* field to ensure backward compatibility, and *d_name* has been moved to the end of the structure. The new code puts the DOS name space name at the *d_nameDOS* offset so old code will still work. This can all be done with relative ease because CLIB allocates the memory for this structure.

See Also

closedir, readdir

PurgeErasedFile

Permanently deletes a file that has been marked for deletion
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.x, 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwfile.h>

int PurgeErasedFile (
   char  *pathname,
   long   sequenceNumber);
```

Parameters

pathname

(IN) Specifies the string containing either the absolute path (including the volume name) or the relative path name of the file to purge (maximum 255 characters, including the NULL terminator).

sequenceNumber

(IN) Identifies which version of the specified file to purge.

Return Values

0	0x0 0	ESUCCESS
NetWare Error		UNSUCCESSFUL

Remarks

An application marks a file for deletion with the **remove** or **unlink** function. However, the server does not permanently delete a file until the server needs the disk space occupied by the file marked for deletion. A file marked for deletion with the remove or unlink functions can be recovered by calling the **SalvageErasedFile** function.

PurgeErasedFile permanently deletes a file marked for deletion. It frees

the disk space that the deleted file occupied. A file deleted with **PurgeErasedFile** cannot be recovered.

NOTE: The *sequenceNumber* parameter must be obtained from the **ScanErasedFiles** function. The current connection must have Delete rights to the file.

There is no need to call the **ScanErasedFiles** function to get a sequence number on remote 286 servers. **PurgeErasedFile** can be called without regard to the validity of the path name or sequence number on 286 servers. 286 servers do not retain files that have been marked for deletion but not yet purged. **PurgeErasedFile** purges all files that have been deleted recently (since the last file system operation.

The **SetCurrentNameSpace** function sets the name space that is used for parsing the path input to **PurgeErasedFile**.

NOTE: **PurgeErasedFile** currently works only in the DOS name space. However, you can purge a file in other name spaces in the following way. Call the **SetCurrentNameSpace** function to change to the DOS name space and then call the **ScanErasedFiles** function to get the DOS names of the files you want to purge. These are returned in the structure that the **ScanErasedFiles** function uses. You can then purge the files, supplying their DOS names as specified by the *pathname* parameter.

See Also

SalvageErasedFile, **ScanErasedFiles**

readdir

Obtains information about the next matching file
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <dirent.h>

DIR *readdir (
   DIR  *dirP);
```

Parameters

dirP

(IN/OUT) Specifies the structure to receive information about the next matching file.

Return Values

Returns a pointer to an object of the DIR structure type containing information about the next matching file or directory.

If an error occurs, such as when there are no more matching file names, NULL is returned and errno and NWErrno are set. (Unless NULL is returned, ignore values in errno and NWErrno.)

Remarks

readdir can be called repeatedly to obtain the list of file and directory names contained in the directory specified by the path name given to the **opendir** function.

The **closedir** function must be called to close the directory and free the memory allocated by the **opendir** function.

NOTE: The date and time fields are not in the DOS date/time format. It is easily put in the DOS format by swapping the high word with the low word.

Beginning with Release 9 of the NW SDK, **readdir** returns long names in the *d_name* field of the dirent structure if the target namespace is

previously set (by calling the **SetTargetNameSpace** function) to something other than DOS. You must compile with the dirent.h file included with Release 9 or later and link with the new nwpre.obj and is valid only when calling **readdir** on the local server.

NOTE: The position in the old structure of the*d_name* field has been assumed by the new *d_nameDOS* field to ensure backward compatibility, and the *d_name* field has been moved to the end of the structure. The new code puts the DOS name space name at the *d_nameDOS* field offset so old code will still work. This can all be done with relative ease because CLIB allocates the memory.

See Using readdir(): Example.

See Also

closedir, **opendir**

remove

Deletes a specified file
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
Platform: NLM
SMP Aware: Yes
Service: File System

Syntax

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

int remove (
   const char  *filename);
```

Parameters

filename

(IN) Specifies the string containing the full or relative path of the file to be deleted (maximum 255 characters, including the NULL terminator).

Return Values

Returns a value of 0 if successful, nonzero otherwise. When an error has occurred, errno contains a value indicating the type of error that has been detected.

Remarks

remove also works on the DOS partition.

remove causes a file to be marked for deletion. A file marked for deletion is not actually erased until the space it occupies is needed by another file. The current connection must have Delete rights to the file.

Wildcard specifiers are allowed for the *filename* parameter.

The **SalvageErasedFile** function can be used to salvage a file that has been marked for deletion but not yet purged.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to **remove**.

NOTE: For NetWare versions before 4.x, **remove** only works with DOS

name space for remote servers.

See Also

PurgeErasedFile, **SalvageErasedFile**, **unlink**

rename

Renames a specified file
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
Platform: NLM
SMP Aware: Yes
Service: File System

Syntax

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

int rename (
   const char  *old,
   const char  *new);
```

Parameters

old

(IN) Points to a string containing the full or relative path of the name of the file to be renamed (maximum 255 characters, including the NULL terminator).

new

(IN) Points to a string containing the full or relative path of the new file name to replace the old file name (maximum 255 characters, including the NULL terminator).

Return Values

Returns a value of 0 if successful, nonzero otherwise.

Remarks

rename also works on the DOS partition.

Wildcard specifiers are allowed for the *old* and *new* parameters.

rename can also rename directories. However, if a wildcard is specified, only matching files (not directories) are renamed.

The current connection number must have Modify privileges. If a wildcard is specified, the current connection must also have See File rights. To move a file, the current connection must have Delete and Read

rights for the file to be moved and Create rights in the destination.

To move a directory requires Delete rights to the directory to be moved and Create in the destination. The above-mentioned rights are also required for all directories and files in the subdirectory tree. Additionally, Create, See File, and Read rights are required to move deleted files; without these rights, deleted files are purged.

NOTE: **rename** only works with DOS name space. However, the **NWSetNameSpaceEntryName** function can rename files in other name spaces.

See Also

FileServerFileCopy, **NWGetNameSpaceEntryName**, **NWSetNameSpaceEntryName**

rmdir

Removes (deletes) the specified directory
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <unistd.h>

int rmdir (
   const char  *pathname);
```

Parameters

pathname

(IN) Specifies either the absolute or relative directory path containing the directory to delete.

Return Values

Returns a value of 0 if successful, nonzero otherwise. If an error occurs, errno and NetWareErrno are set.

Remarks

rmdir also works on the DOS partition.

The directory must not contain any files or directories.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to **rmdir**.

NOTE: For NetWare versions before 4.x, **rmdir** only works with DOS name space for remote servers.

See Also

chdir, **getcwd**, **mkdir**

SalvageErasedFile

Salvages a file that has been marked for deletion
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.x, 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwfile.h>

int SalvageErasedFile (
   char  *pathname,
   long   sequenceNumber,
   char  *newFileName);
```

Parameters

pathname

(IN) Specifies the string containing the path name of the erased file to be salvaged (maximum 255 characters, including the NULL terminator).

sequenceNumber

(IN) Specifies which version of the specified file to restore.

newFileName

(IN) Points to a NULL-terminated string containing the name to give the erased file when it is restored (maximum 13 characters, including the NULL terminator).

See Salvaging Files: Example.

Return Values

0	0x0 0	ESUCCESS
NetWare Error		UNSUCCESSFUL

Remarks

A file marked for deletion with the **remove** or **unlink** function can be recovered by calling the **SalvageErasedFile** function.

The *pathname* parameter can be an absolute path with a volume name, or it can be relative to the current working directory. In a NetWare 2.x environment, only the volume name should be passed in the path, not the full path name.

The *sequenceNumber* parameter is obtained from the **ScanErasedFiles** function.

The *newFileName* parameter can be from 1 to 8 characters long and can also include an extension of from 1 to 3 characters. All letters must be uppercase and the string must be NULL-terminated.

The current connection must have Create rights in the specified directory.

The **SetCurrentNameSpace** function sets the name space that is used for parsing the path input to **rmdir**.

NOTE: **rmdir** currently works only in the DOS name space. However, you can salvage a file in other name spaces in the following way. Call the **SetCurrentNameSpace** function to change to the DOS name space. Then call the **ScanErasedFiles** function to get the DOS names of the files you want to salvage. The DOS names are returned in the structure that the **ScanErasedFiles** function uses. You can then salvage the files, supplying their DOS names to the *pathname* parameter. After you have salvaged the files, they still have directory entries in the other name spaces that are loaded just as they did before they were deleted.

See Also

PurgeErasedFile, **ScanErasedFiles**

ScanErasedFiles

Returns information about deleted files
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwfile.h>

int ScanErasedFiles (
   char  *pathname,
   long  *nextEntryNumber,
   DIR   *deletedFileInfo);
```

Parameters

pathname

(IN) Specifies the string containing the path specification of the directory to view (maximum 255 characters, including the NULL terminator).

nextEntryNumber

(IN/OUT) Points to the entry number of the next file (-1 initially).

deletedFileInfo

(OUT) Points to the DIR structure.

Return Values

0	0x00	ESUCCESS
NetWare Error		UNSUCCESSFUL

Remarks

ScanErasedFiles can be called repeatedly to obtain the list of file names contained in the directory specified by the *pathname* parameter. Files marked for deletion can be scanned to obtain information about who deleted the files and when they were deleted.

The *pathname* parameter can be an absolute path with a volume name or it can be relative to the current working directory. Do not include a wildcard character at the end of the path. In the following example, the erased files in the DIR1 directory on the SYS volume are scanned:

```
SYS:DIR1
```

The *nextEntryNumber* parameter returns an entry number for the next matching file name.

The *deletedFileInfo* parameter contains information about the matching file name.

The current connection must have See File rights in the specified directory.

The **SetCurrentNameSpace** function sets the name space that is used for parsing the path input to **ScanErasedFiles**.

NOTE: **ScanErasedFiles** currently works only in the DOS name space. However, you can scan erased files for another name space. Call the **SetCurrentNameSpace** function to change to the DOS name space. Then call **ScanErasedFiles**, supplying a DOS path name.

ScanErasedFiles returns DOS names for the files that have been erased. You can then use those names to either salvage the files by calling the **SalvageErasedFile** function or purge them by calling the **PurgeErasedFile** function.

See Also

PurgeErasedFile, **SalvageErasedFile**

SetExtendedFileAttributes

Sets the extended attributes byte for a file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwfile.h>

int SetExtendedFileAttributes (
   char  *filePath,
   BYTE   extendedFileAttributes);
```

Parameters

filePath

(IN) Specifies the string containing the relative or absolute (including the volume name) path specification of the file whose extended attributes are being changed (maximum 255 characters, including the NULL terminator).

extendedFileAttributes

(IN) Specifies the new extended attributes for the file.

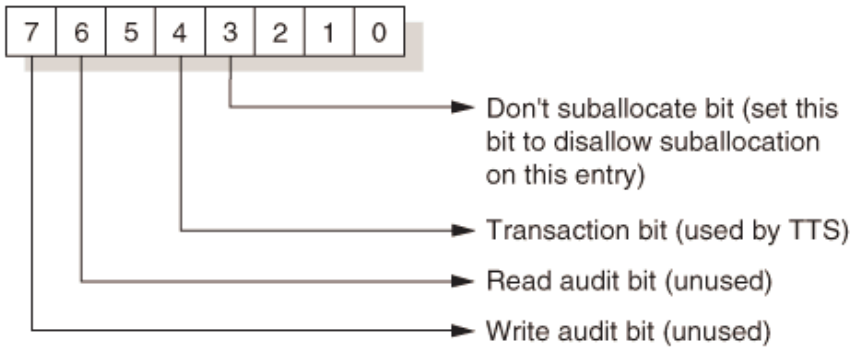
Return Values

0	0x00	ESUCCESS
254	0xFE	ERR_INCORRECT_ACCESS_PRIVILEGES
255	0xFF	ERR_NO_FILES_FOUND

Remarks

SetExtendedFileAttributes sets the extended file attributes for a file by passing a file path and an extended file attributes byte. The current connection must have Modify rights to the file.

SetExtendedFileAttributes overwrites the first byte of the existing file attributes with the value in the *extendedFileAttributes* parameter. The byte definition follows:



If the Transaction bit is set in the *extendedFileAttributes* parameter byte, TTS tracks all writes to the file during a transaction. A transaction file cannot be deleted or renamed until the transaction bit is turned off by calling **SetExtendedFileAttributes**.

NOTE: Do not confuse the first attributes byte with true extended attributes, which can be manipulated by calling the Extended Attribute functions.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to **SetExtendedFileAttributes**.

NOTE: For NetWare versions before 4.x, **SetExtendedFileAttributes** only works with DOS name space for remote servers.

See Also

GetExtendedFileAttributes

SetFileInfo

Sets file information for a file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.x, 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <nwfile.h>

int SetFileInfo (
   char  *filePath,
   BYTE   searchAttributes,
   LONG   fileAttributes,
   char  *creationDateAndTime,
   char  *lastAccessDate,
   char  *lastUpdateDateAndTime,
   char  *lastArchiveDateAndTime,
   LONG   fileOwnerID);
```

Parameters

filePath

(IN) Points to the string containing the path specification of the file to be changed (maximum 255 characters, including the NULL terminator).

searchAttributes

(IN) Specifies the type of the file for which to set file information.

fileAttributes

(IN) Specifies the file attributes to be assigned to the file.

creationDateAndTime

(IN) Points to the creation date and time to be assigned to the file (DOS format, 4 bytes).

lastAccessDate

(IN) Points to the last access date to be assigned to the file (DOS format, bytes 1 and 2).

lastUpdateDateAndTime

(IN) Points to the last update date and time to be assigned to the file (DOS format, 4 bytes).

lastArchiveDateAndTime

(IN) Points to the last archived date and time to be assigned to the file (DOS format, 4 bytes).

fileOwnerID

(IN) Specifies the unique object ID to be assigned as the new owner.

Return Values

0	0x00	ESUCCESS
NetWare Error		UNSUCCESSFUL

Remarks

SetFileInfo sets file information by passing the file path, the search attributes byte, and specific file information. File information includes file attributes, extended file attributes, creation date and time, last access date, last update date and time, file owner, and last archived date and time.

SetFileInfo requires that the requesting workstation have Supervisor rights to the file(s) being modified.

The *filePath* parameter can specify an absolute or a relative path. An absolute file path appears in the following format:

```
volume: directory1\...\directory\file name
```

A relative file path includes a file name and (optionally) one or more antecedent directory names.

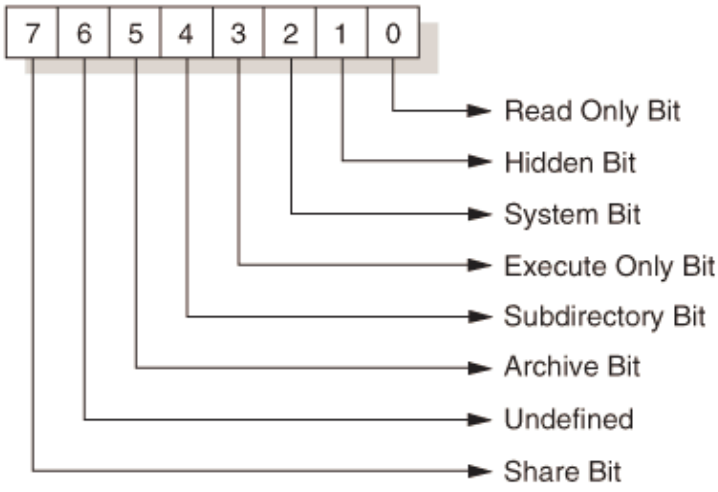
A file name can be from 1 to 8 characters long and can include a 1- to 3-character extension. All letters must be upper case. The last item in the *filePath* parameter must be a valid file name specification. No wildcard specifiers are allowed.

The *searchAttributes* parameter can have the following values:

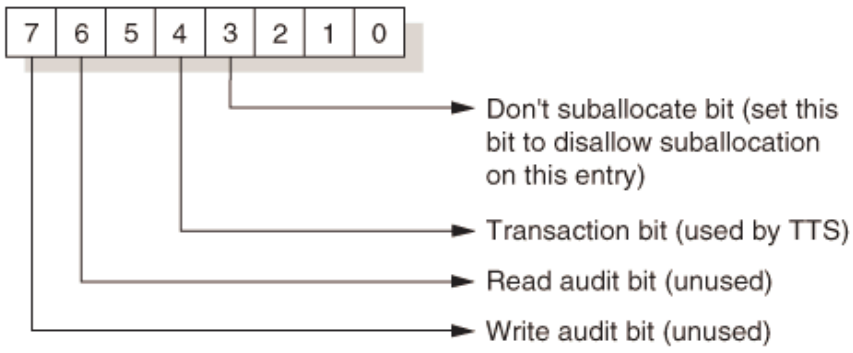
0x00	Normal files
0x02	Normal and hidden files
0x04	Normal and system files

0x 06	Normal, hidden, and system files

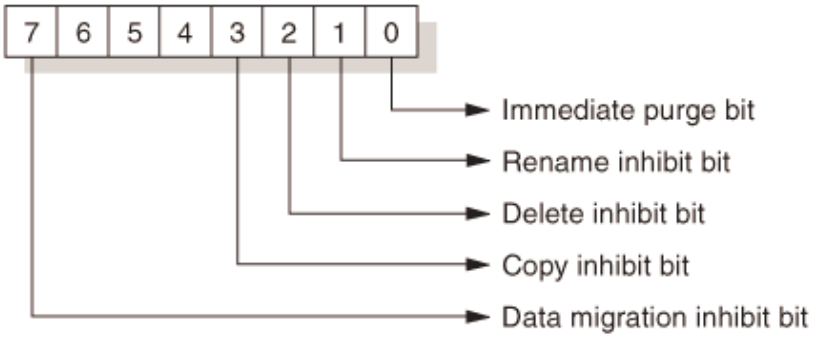
SetFileInfo can assign file attributes to a specified file by passing a new value in the *fileAttributes* parameter. The following bits are defined for byte 0:



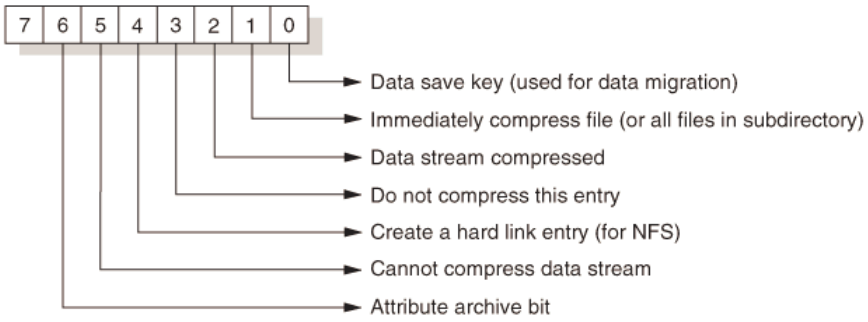
The following bits are defined for byte 1, the extended attributes byte:



In NetWare 3.0 and above, you can set four file attributes in byte 2, bits 0, 1, 2, and 4. In NetWare 4.x, you can set bit 7:



NetWare 4.x also allows you to set file attributes in an additional byte, byte 3:



The *creationDateAndTime*, *lastUpdateDateAndTime*, and *lastArchiveDateAndTime* parameters occupy bytes 0, 1, 2, and 3.

The application can change the owner of the file by passing the object ID number of the new owner in the *fileOwnerID* parameter.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to **SetFileInfo**.

See Also

NWSetDirEntryInfo, **readdir**

SetReaddirAttribute

Sets the attributes that are to be used when searching for files and directories by calling the **readdir** function

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: NLM

SMP Aware: No

Service: File System

Syntax

```
#include <nwfile.h>

int SetReaddirAttribute (
   DIR             *dirP,
   unsigned long    newAttribute);
```

Parameters

dirP

(IN) Points to the DIR structure obtained by calling **opendir** or **readdir**.

newAttribute

(IN) Specifies the new attribute.

Return Values

Returns a value of 0 if successful, nonzero otherwise.

Remarks

SetReaddirAttribute can be called any time after the DIR structure has been obtained from the **opendir** function. The modified search attributes are in effect for calling the **readdir** function.

The following search attributes are defined:

_A_NORMAL	Normal file; read/write permitted
_A_RDONLY	Read-only file
_A_HIDDEN	Hidden file

_A_SYSTEM	System file
_A_VOLID	Volume ID entry
_A_SUBDIR	Subdirectory
_A_ARCH	Archive file

See Also

closedir, opendir, readdir

_splitpath

Splits a full path name into four components consisting of a server/volume name, directory path, file name, and file name extension

Local Servers: blocking

Remote Servers: N/A

Platform: NLM

SMP Aware: No

Service: File System

Syntax

```
#include <nwdir.h>

void _splitpath (
   const char  *path,
   char        *drive,
   char        *dir,
   char        *fname,
   char        *ext);
```

Parameters

path

(IN) Specifies the string containing the full path name to split.

drive

(OUT) Points to the server/volume name or drive letter.

dir

(OUT) Points to the directory path.

fname

(OUT) Points to the base name of the file without an extension.

ext

(OUT) Points to the file name extension, including the leading period.

Return Values

None

Remarks

_splitpath returns the drive letter in the*drive* parameter. If you pass it a NetWare path, **_splitpath** returns the NetWare server/volume in the *drive* parameter.

The *drive*, *dir*, *fname*, and *ext* parameters are not filled in if they are NULL. For each component of the full path name that is not present, its corresponding buffer is set to an empty string.

See Using _makepath and _splitpath: Example.

See Also

_makepath

stat (Function)

Retrieves the status of a specified file or directory
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <stat.h>

int stat (
   const char   *path,
   struct stat  *statblk);
```

Parameters

path

(IN) Points to a string containing the path of the directory or file for which status is to be obtained (maximum 255 characters, including the NULL terminator).

statblk

(OUT) Points to the stat (Structure) containing information about the file.

Return Values

Returns a value of 0 when the information is successfully obtained. Otherwise, a value of -1 is returned and errno is set to indicate the type of error that occurred.

Remarks

stat (Function) returns information in the stat (Structure) located at the address indicated by the *statblk* parameter.

The SYS\STAT.H header file contains definitions for the stat (Structure) and describes the contents of the fields.

The time and date in the stat (Structure) are in calendar format.

Beginning with Release 9 of the NW SDK, **stat (Function)** returns long names in the *d_name* field of the stat (Structure) if the *st_name* field is set to

something other than DOS. You must compile with the stat.h file included with Release 9 or later and link with the new nwpre.obj and is valid only when calling **stat (Function)** on the local server.

The current connection must have See File rights.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to **stat (Function)**.

NOTE: For NetWare versions before 4.x, **stat (Function)** only works with DOS name space for remote servers.

See Also

fstat

tmpnam

Generates a unique string for use as a valid temporary file name
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
Platform: NLM
SMP Aware: Yes
Service: File System

Syntax

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

char *tmpnam (
   char  *buffer);
```

Parameters

buffer

(OUT) Points to the buffer to receive the generated temporary file name.

Return Values

If you pass a NULL pointer, **tmpnam** leaves the temporary file name in an internal static buffer and returns a pointer to that buffer.

Remarks

Be aware that the internal static buffer is modified every time **tmpnam** is called, whether or not you pass a NULL pointer. If you want to preserve the temporary file name currently stored in the internal static buffer, copy it to another buffer (by calling the **strcpy** function) before calling **tmpnam** again.

If you pass a pointer to your created array, **tmpnam** leaves the temporary file name in that array and returns a pointer to it. **tmpnam** simply returns the pointer you have supplied. It does no error checking to ensure that your array is big enough to accommodate the file name. The array should be at least L_tmpnam characters in length, where L_tmpnam is 13 characters (12 for the DOS 8.3 characters plus one for the NULL terminator).

See Using tmpnam(): Example.

See Also

access

umask

Sets the file permission mask (part of the thread group context).
Local Servers: blocking
Remote Servers: N/A
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <stat.h>

int umask (
   int   permission);
```

Parameters

permission

(IN) Specifies the file permission mask to be used to update the permission of the current process.

Return Values

Returns the previous value of the *permission* parmeter.

Remarks

The file permission mask is used to modify the permission setting of new files created by the **creat**, **open**, or **sopen** function. If a bit in the mask is on, the corresponding bit in the requested permission value for the file is disallowed.

The *permission* parameter is a constant expression involving the constants S_IREAD and S_IWRITE as defined in SYS\STAT.H.

S_IWRITE	Write permission
S_IREAD	Read permission

See Also

chmod, **creat**, **open**, **sopen**

unlink

Deletes the specified file
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <unistd.h>

int unlink (
   const char  *filename);
```

Parameters

filename

(IN) Points to a string containing the absolute or relative path of the file to delete (maximum 255 characters, including the NULL terminator).

Return Values

Returns a value of 0 if successful, nonzero otherwise. When an error has occurred, errno contains a value indicating the type of error that has been detected.

Remarks

unlink also works on the DOS partition.

A file marked for deletion is not actually erased by **unlink** until the space it occupies is needed by another file.

Wildcard specifiers are allowed for the *filename* parameter.

The **SalvageErasedFile** function can be called to salvage a file that has been marked for deletion but not yet purged.

The current connection must have Delete rights to the file.

See Using unlink(): Example.

See Also

PurgeErasedFile, remove, SalvageErasedFile

utime

Updates the modification time for the specified file
Local Servers: blocking
Remote Servers: blocking
Platform: NLM
SMP Aware: No
Service: File System

Syntax

```
#include <sys\utime.h>

int utime (
   const char      *filename,
   struct utimbuf  *times);
```

Parameters

filename

(IN) Points to a string containing the name of the file whose modification time is to be updated (maximum 255 characters, including the NULL terminator).

times

(IN) Points to the structure containing the modification time.

Return values

Returns a value of 0 when the time was successfully recorded. A value of -1 indicates an error occurred. If an error occurs, errno is set.

Remarks

If the *filename* parameter specifies a directory, the modification time and date are updated and the last accessed date is ignored (since directories do not have a last accessed date).

If the *times* parameter is NULL, the current time is used for the update. Otherwise, the *times* parameter must point to an object of the struct utimbuf type.

The modification time is taken from the *modtime* field in the structure and the last accessed date is taken from the *acctime* field.

The current connection must have Modify rights or Write rights to update the last modification time. It must also have Modify or Read rights to

update the last accessed date.

The **SetCurrentNameSpace** function sets the name space which is used for parsing the path input to **utime**.

NOTE: For NetWare versions before 4.x, **utime** only works with DOS name space for remote servers.

File System: Structures

CONN_USING_FILE

Defines file information for a file opened by a connection
Service: File System
Defined In: nwfile.h

Structure

```
typedef struct {
   NWCONN_NUM   connNumber;
   nuint16      taskNumber;
   nuint8       lockType;
   nuint8       accessControl;
   nuint8       lockFlag;
} CONN_USING_FILE;
```

Pascal Structure

```
Defined in nwfile.inc

 CONN_USING_FILE = Record
    connNumber : NWCONN_NUM;
    taskNumber : nuint16;
    lockType : nuint8;
    accessControl : nuint8;
    lockFlag : nuint8
  End;
```

Fields

connNumber

Specifies the logical connection number of a workstation using the file.

taskNumber

Specifies the number of the task which opened the file. A given connection may have several task numbers associated with the same file.

lockType

Specifies how the file is locked.

accessControl

Specifies how the file is accessed.

lockFlag

Specifies whether the file is locked.

Remarks

The *lockType* field can have the following values:

0x01 Locked
0x02 Open shareable
0x04 Logged
0x08 Open Normal
0x40 TTS holding
0x80 Transaction flag set

The *accessControl* field can have the following values:

0x01 Open for read by this client
0x02 Open for write by this client
0x04 Deny read requests from others
0x08 Deny write requests from others
0x10 File detached
0x20 TTS holding detach
0x40 TTS holding open

The *lockFlag* field can have the following values:

0x00 Not locked
0xFE Locked by a file lock
0xFF Locked by begin share file set

CONNS_USING_FILE

Returns a list of connections having a specified file open
Service: File System
Defined In: nwfile.h

Structure

```
typedef struct {
   nuint16           nextRequest;
   nuint16           useCount;
   nuint16           openCount;
   nuint16           openForReadCount;
   nuint16           openForWriteCount;
   nuint16           denyReadCount;
   nuint16           denyWriteCount;
   nuint8            locked;
   nuint8            forkCount;
   nuint16           connCount;
   CONN_USING_FILE   connInfo[70];
} CONNS_USING_FILE;
```

Pascal Structure

```
Defined in nwfile.inc

CONNS_USING_FILE = Record
    nextRequest : nuint16;
    useCount : nuint16;
    openCount : nuint16;
    openForReadCount : nuint16;
    openForWriteCount : nuint16;
    denyReadCount : nuint16;
    denyWriteCount : nuint16;
    locked : nuint8;
    forkCount : nuint8;
    connCount : nuint16;
    connInfo : Array[0..69] Of CONN_USING_FILE
  End;
```

Fields

nextRequest

Specifies the sequence in subsequent calls to the **NWScanConnectionsUsingFile** function.

useCount

Specifies the number of tasks having the file opened or logged.

openCount

Specifies the number of tasks having opened or logged the file.

openForReadCount

Specifies the number of logical connections having the file open for reading.

openForWriteCount

Specifies the number of logical connections having the file open for writing.

denyReadCount

Specifies the number of logical connections having denied other connections access to the file.

denyWriteCount

Specifies the number of logical connections having denied other connections read access to the file.

locked

Specifies whether the file is locked exclusively (0=not locked exclusively).

forkCount

Specifies the number of forks associated with the file.

connCount

Specifies the number of connections using the file.

connInfo

Specifies an array of CONN_USING_FILE structures specifying how each connection is using the file.

DIR

Holds information about a directory entry
Service: File System
Defined In: dirent.h

Structure

```
typedef struct dirent {
   unsigned long    d_attr;
   unsigned short   d_time;
   unsigned short   d_date;
   long             d_size;
   ino_t            d_ino;
   dev_t            d_dev;
   unsigned long    d_cdatetime;
   unsigned long    d_adatetime;
   unsigned long    d_bdatetime;
   long             d_uid;
   unsigned long    d_archivedID;
   unsigned long    d_updatedID;
   char             d_nameDOS[13];
   unsigned short   d_inheritedRightsMask;
   unsigned char    d_originatingNameSpace;
   unsigned long    d_ddatetime;
   unsigned long    d_deletedID;
   char             d_name[255+1];

} DIR;
```

Fields

d_attr
Specifies the attribute as defined in NWFATTR.H.

d_time
Specifies the modification time in DOS format.

d_date
Specifies the modification date in DOS format.

d_size
Specifies the size (files only).

d_ino
Specifies the serial number.

d_dev
Specifies the volume number.

d_cdatetime

Specifies the creation date and time in DOS format.

d_adatetime

Specifies the last access date (files only) in DOS format.

d_bdatetime

Specifies the last archive date and time in DOS format.

d_uid

Specifies the owner ID (object ID).

d_archivedID

Specifies the object ID that last archived the file.

d_updateID

Specifies the object ID that last updated the file.

d_nameDOS

Specifies the DOS name space name.

d_inheritedRightsMask

Specifies the inherited rights mask.

d_originatingNameSpace

Specifies the creating name space.

d_ddatetime

Specifies the date and time the entry was deleted (used by the **ScanErasedFiles** function only).

d_deletedID

Specifies the object ID that deleted the file (used by the **ScanErasedFiles** function only).

d_name

Specifies the namespace name of the entry.

DIR_SPACE_INFO

Returns directory space information
Service: File System
Defined In: nwdirect.h

Structure

```
typedef struct {
   nuint32   totalBlocks;
   nuint32   availableBlocks;
   nuint32   purgeableBlocks;
   nuint32   notYetPurgeableBlocks;
   nuint32   totalDirEntries;
   nuint32   availableDirEntries;
   nuint32   reserved;
   nuint8    sectorsPerBlock;
   nuint8    volLen;
   nuint8    volName[NW_MAX_VOLUME_NAME_LEN];
} DIR_SPACE_INFO;
```

Pascal Structure

```
Defined in nwdirect.inc

 DIR_SPACE_INFO = Record
   totalBlocks : nuint32;
   availableBlocks : nuint32;
   purgeableBlocks : nuint32;
   notYetPurgeableBlocks : nuint32;
   totalDirEntries : nuint32;
   availableDirEntries : nuint32;
   reserved : nuint32;
   sectorsPerBlock : nuint8;
   volLen : nuint8;
   volName : Array[0..NW_MAX_VOLUME_NAME_LEN-1] Of nuint8
 End;
```

Fields

totalBlocks

Specifies the total blocks in the directory.

availableBlocks

Specifies the number of available blocks.

purgeableBlocks

Specifies the number of recoverable blocks recovered by purging (0 if

the **NWGetDirSpaceInfo** function is called with a directory handle of 0).

notYetPurgeableBlocks

Specifies the number of blocks not yet purgeable (0 if the **NWGetDirSpaceInfo** function is called with a directory handle of 0).

totalDirEntries

Specifies the number of entries in the directory.

availableDirEntries

Specifies the number of available entries remaining.

reserved

Is reserved.

sectorsPerBlock

Specifies the number of sectors per block.

volLen

Specifies the length of the *volName* field.

volName

Specifies the name of the volume.

GrammarTableStruct

Service: File System
Defined In: nwconfig.h

Structure

```
typedef struct {
   int                 keywordCode;
   char N_FAR         *keyword;
   void (N_FAR        *function)(PARAMETER_TABLE_TYPE N_FAR *);
   TypeDefaultStruct   typeDefault[MAX_PARAMETERS];
} GrammarTableStruct;
```

Fields

keywordCode

Specifies the message number for the keyword in the MSG file.

keyword

Points to a character array containing the name of the key word being searched for within the net.cfg section (in the Link Support Section, keywords could be BUFFERS or MAX STACKS).

function

Specifies the function to be called when a match is found.

typeDefault

Specifies the array containing the types of values and default values of optional types in which you are interested.

Remarks

The function specified by the *function* field is called with one parameter. The parameter is a pointer to the PARAMETER_TABLE_TYPE structure which should be cast to the value you expect to receive.

The *typeDefault* field can have the following values:

01h T_NUMBER
02h T_INDEX
03h T_STRING
04h T_HEX_STRING
05h T_HEX_NUMBER
06h T_LONG_NUMBER
07h T_LONG_HEX
10h T_SET_1
11h T_SET_2

12h T_SET_3
13h T_SET_4
14h T_SET_5
15h T_SET_6
16h T_SET_7
17h T_SET_8
18h T_SET_9
19h T_SET_10
80h T_OPTIONAL
1Ah T_SET_11
1Bh T_SET_12
1Ch T_SET_13
1Dh T_SET_14
1Eh T_SET_15
1Fh T_SET_16

ModifyStructure

Holds information used in changing a directory entry
Service: File System
Defined In: nwdir.h

Structure

```
typedef struct {
   BYTE   *MModifyName;
   LONG   MFileAttributes;
   LONG   MFileAttributesMask;
   WORD   MCreateDate;
   WORD   MCreateTime;
   LONG   MOwnerID;
   WORD   MLastArchivedDate;
   WORD   MLastArchivedTime;
   LONG   MLastArchivedID;
   WORD   MLastUpdatedDate;
   WORD   MLastUpdatedTime;
   LONG   MLastUpdatedID;
   WORD   MLastAccessedDate;
   WORD   MInheritanceGrantMask;
   WORD   MInheritanceRevokeMask;
   int    MMaximumSpace;
   LONG   MLastUpdatedInSeconds;
} ModifyStructure;
```

Fields

MModifyName

Specifies the new directory name.

MFileAttributes

Specifies new file attributes.

MFileAttributesMask

Specifies new file attribute mask.

MCreateDate

Specifies new creation date.

MCreateTime

Specifies new creation time

MOwnerID

Specifies new owner ID.

MLastArchivedDate

Specifies the last archived date.

MLastArchivedTime

Specifies the last archived time.

MLastArchivedID

Specifies the last archived ID.

MLastUpdatedDate

Specifies the last updated date.

MLastUpdatedTime

Specifies the last updated time.

MLastUpdatedID

Specifies the last updated ID.

MLastAccessedDate

Specifies the last accessed date.

MInheritanceGrantMask

Specifies the inheritance grant mask.

MInheritanceRevokeMask

Specifies in heritance revoke mask.

MMaximumSpace

Specifies the maximum space.

MLastUpdatedInSeconds

Specifies the last update in seconds.

NW_EXT_FILE_INFO

Returns extended file information
Service: File System
Defined In: nwdentry.h

Structure

```
typedef struct {
   nuint32   sequence;
   nuint32   parent;
   nuint32   attributes;
   nuint8    uniqueID;
   nuint8    flags;
   nuint8    nameSpace;
   nuint8    nameLength;
   nuint8    name[12];
   nuint32   creationDateAndTime;
   nuint32   ownerID;
   nuint32   lastArchiveDateAndTime;
   nuint32   lastArchiverID;
   nuint32   updateDateAndTime;
   nuint32   lastUpdatorID;
   nuint32   dataForkSize;
   nuint32   dataForkFirstFAT;
   nuint32   nextTrusteeEntry;
   nuint8    reserved[36];
   nuint16   inheritedRightsMask;
   nuint16   lastAccessDate;
   nuint32   deletedFileTime;
   nuint32   deletedDateAndTime;
   nuint32   deletorID;
   nuint8    reserved2[16];
   nuint32   otherForkSize[2];
} NW_EXT_FILE_INFO;
```

Pascal Structure

```
NW_EXT_FILE_INFO = Record
  sequence : nuint32;
  parent : nuint32;
  attributes : nuint32;
  uniqueID : nuint8;
  flags : nuint8;
  nameSpace : nuint8;
  nameLength : nuint8;
  name : Array[0..11] Of nuint8;
  creationDateAndTime : nuint32;
  ownerID : nuint32;
```

```
    lastArchiveDateAndTime : nuint32;
    lastArchiverID : nuint32;
    updateDateAndTime : nuint32;
    lastUpdatorID : nuint32;
    dataForkSize : nuint32;
    dataForkFirstFAT : nuint32;
    nextTrusteeEntry : nuint32;
    reserved : Array[0..35] Of nuint8;
    inheritedRightsMask : nuint16;
    lastAccessDate : nuint16;
    deletedFileTime : nuint32;
    deletedDateAndTime : nuint32;
    deletorID : nuint32;
    reserved2 : Array[0..15] Of nuint8;
    otherForkSize : Array[0..1] Of nuint32
  End;
```

Fields

sequence

Specifies the sequence for iteratively scanning entries (-1 initially).

parent

Specifies the directory entry ID of parent directory.

attributes

Specifies the attributes of the entry.

uniqueID

Specifies the unique entry ID.

flags

Is reserved.

nameSpace

Specifies the name space creating the entry.

nameLength

Specifies the maximum number of characters in the name.

name

Specifies the entry name.

creationDateAndTime

Specifies when the entry was created.

ownerID

Specifies the object ID of the owner.

lastArchiveDateAndTime

Specifies when the entry was last archived.

lastArchiverID

Specifies the ID of the object last archiving the entry.

updateDateAndTime

Specifies the date and time when the entry was last modified.

lastUpdatorID

Specifies the ID of the object that last modified the entry.

dataForkSize

Specifies the number of bytes in the file.

dataForkFirstFAT

nextTrusteeEntry

Specifies the next trustee of the entry.

reserved

Is reserved.

inheritedRightsMask

Specifies the Inherited Rights Mask for the entry.

lastAccessDate

Specifies the date when the entry was last accessed.

deletedFileTime

Specifies the time when the file was deleted.

deletedDateAndTime

Specifies the date and time when the entry was deleted.

deletorID

Specifies the ID of the object deleting the entry.

reserved2

Is reserved.

otherForkSize

Remarks

The *attributes* field can have the following values:

C Value	Pascal Value	Value Name
0x00000000L	$00000000	A_NORMAL
0x00000001L	$00000001	A_READ_ONLY
0x00000002L	$00000002	A_HIDDEN

0x00000004L	$00000004	A_SYSTEM
0x00000008L	$00000008	A_EXECUTE_ONLY
0x00000010L	$00000010	A_DIRECTORY
0x00000020L	$00000020	A_NEEDS_ARCHIVED
0x00000080L	$00000080	A_SHAREABLE
0x00001000L	$00000100	A_TRANSACTIONAL
0x00002000L	$00000200	A_INDEXED
0x00004000L	$00000400	A_READ_AUDIT
0x00008000L	$00000800	A_WRITE_AUDIT
0x00010000L	$00001000	A_IMMEDIATE_PURGE
0x00020000L	$00002000	A_RENAME_INHIBIT
0x00040000L	$00004000	A_DELETE_INHIBIT
0x00080000L	$00008000	A_COPY_INHIBIT
0x00400000L	$00400000	A_FILE_MIGRATED
0x00800000L	$00800000	A_DONT_MIGRATE
0x02000000L	$02000000	A_IMMEDIATE_COMPRESS
0x04000000L	$04000000	A_FILE_COMPRESSED
0x08000000L	$08000000	A_DONT_COMPRESS
0x20000000L	$20000000	A_CANT_COMPRESS

The *nameSpace* field can have the following values:

0 NW_NS_DOS
1 NW_NS_MAC

2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_OS2
4 NW_NS_LONG

The *inheritedRightsMask* field can have the following values:

C Value	Pascal Value	Value Description
0x0000	$0000	TR_NONE
0x0001	$0001	TR_READ
0x0002	$0002	TR_WRITE
0x0004	$0004	TR_OPEN
0x0004	$0004	TR_DIRECTORY
0x0008	$0008	TR_CREATE
0x0010	$0010	TR_DELETE
0x0010	$0010	TR_ERASE
0x0010	$0020	TR_OWNERSHIP
0x0020	$0020	TR_ACCESS_CTRL
0x0040	$0040	TR_FILE_SCAN
0x0040	$0040	TR_SEARCH
0x0040	$0040	TR_FILE_ACCESS
0x0080	$0080	TR_MODIFY
0x01FB	$01FB	TR_ALL
0x0100	$0100	TR_SUPERVISOR
0x00F	$00F	TR_NORMAL


B
B

NW_FILE_INFO2

Holds file information
Service: File System
Defined In: nwfile.h

Structure

```
typedef struct {
   nuint8    fileAttributes;
   nuint8    extendedFileAttributes;
   nuint32   fileSize;
   nuint16   creationDate;
   nuint16   lastAccessDate;
   nuint32   lastUpdateDateAndTime;
   nuint32   fileOwnerID;
   nuint32   lastArchiveDateAndTime;
   nstr8     fileName[260];
} NW_FILE_INFO2;
```

Pascal Structure

```
Defined in nwfile.inc

NW_FILE_INFO2 = Record
    fileAttributes : nuint8;
    extendedFileAttributes : nuint8;
    fileSize : nuint32;
    creationDate : nuint16;
    lastAccessDate : nuint16;
    lastUpdateDateAndTime : nuint32;
    fileOwnerID : nuint32;
    lastArchiveDateAndTime : nuint32;
    fileName : Array[0..259] Of nstr8
  End;
```

Fields

fileAttributes

Specifies the following file attributes.

extendedFileAttributes

Specifies the following file extended attributes:

fileSize

Specifies the size of the file.

creationDate

Specifies when the file was created.

Specifies when the file was created.

lastAccessDate

Specifies when the file was last accessed.

lastUpdateDateAndTime

Specifies when the file was last updated.

fileOwnerID

Specifies the object ID of the owner.

lastArchiveDateAndTime

Specifies when the file was last archived.

fileName

Specifies the name of the file (long names are supported).

Remarks

The *fileAttributes* field can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

The *extendedFileAttributes* field can have the following values:

C Value	Pascal Value	Value Name
0x10	$10	FA_TRANSACTIONAL
0x20	$20	FA_INDEXED
0x40	$40	FA_READ_AUDIT
0x80	$80	FA_WRITE_AUDIT

NW_LIMIT_LIST

Returns disk space information about the restrictions along the directory path

Service: File System

Defined In: nwdirect.h

Structure

```
typedef struct {
   nuint8      numEntries;
   struct {
      nuint8   level;
      nuint32  max;
      nuint32  current;
   } list[102];
} NW_LIMIT_LIST
```

Pascal Structure

```
Defined in nwdirect.inc

  NW_LIMIT_LIST = Record
    numEntries : nuint8;
    list : Array[0..101] Of listTag;
  End;
  listTag = Record
    level :  nuint8;
    max : nuint32;
    current : nuint32
  End;
```

Fields

numEntries

Specifies the number of entries returned in the structure.

level

Specifies the distance from the directory to the root for each entry.

max

Specifies the maximum amount of space assigned to a directory for each entry.

current

Specifies the amount of space assigned to a directory minus the amount of space used by a directory and its subdirectories for each entry.

Remarks

If the *max* field for a directory is 0x7FFFFFFF, there is no restriction for the entry. If the *max* field is greater than 0x7FFFFFFF, the limit is zero. The same is true for the *current* field. The *max* and *current* fields are allowed to be negative so a valid space-in-use value may be calculated.

The space-in-use value can be calculated by subtracting the value of the *current* field from the value of the *max* field.

NWDIR_INFO

Defines entry information for directories
Service: File System
Defined In: nwdentry.h

Structure

```
typedef struct {
   nuint32   lastModifyDateAndTime;
   nuint32   nextTrusteeEntry;
   nuint8    reserved[48];
   nuint32   maximumSpace;
   nuint16   inheritedRightsMask;
   nuint8    reserved2[14];
   nuint32   volObjectID;
   nuint8    reserved3[8];
} NWDIR_INFO;
```

Pascal Structure

```
NWDIR_INFO = Record
   lastModifyDateAndTime : nuint32;
   nextTrusteeEntry : nuint32;
   reserved : Array[0..47] Of nuint8;
   maximumSpace : nuint32;
   inheritedRightsMask : nuint16;
   reserved2 : Array[0..13] Of nuint8;
   volObjectID : nuint32;
   reserved3 : Array[0..7] Of nuint8
 End;
```

Fields

lastModifyDateAndTime

Specifies when the directory was last updated.

nextTrusteeEntry

Specifies the next trustee entry in the subdirectory.

reserved

Is reserved.

maximumSpace

Specifies the maximum space available in the subdirectory.

inheritedRightsMask

Specifies the Inherited Rights Mask.

reserved2

Is reserved.

volObjectID

Specifies the volume object ID.

reserved3

Is reserved.

Remarks

The *inheritedRightsMask* field can have the following values:

C Value	Pascal Value	Value Description
0x0000	$0000	TR_NONE
0x0001	$0001	TR_READ
0x0002	$0002	TR_WRITE
0x0004	$0004	TR_OPEN
0x0004	$0004	TR_DIRECTORY
0x0008	$0008	TR_CREATE
0x0010	$0010	TR_DELETE
0x0010	$0010	TR_ERASE
0x0020	$0020	TR_OWNERSHIP
0x0020	$0020	TR_ACCESS_CTRL
0x0040	$0040	TR_FILE_SCAN
0x0040	$0040	TR_SEARCH
0x0040	$0040	TR_FILE_ACCESS
0x0080	$0080	TR_MODIFY

80	0	
0x01FB	$01FB	TR_ALL
0x0100	$0100	TR_SUPERVISOR
0x00FB	$00FB	TR_NORMAL

NWENTRY_INFO

Defines directory entry information
Service: File System
Defined In: nwdentry.h

Structure

```
typedef struct {
   nuint32   sequence;
   nuint32   parent;
   nuint32   attributes;
   nuint8    uniqueID;
   nuint8    flags;
   nuint8    nameSpace;
   nuint8    nameLength;
   nuint8    name[12];
   nuint32   creationDateAndTime;
   nuint32   ownerID;
   nuint32   lastArchiveDateAndTime;
   nuint32   lastArchiverID;
 union {
      NWFILE_INFO   file;
      NWDIR_INFO    dir;
   } info;
} NWENTRY_INFO;
```

Pascal Structure

```
Defined in nwdentry.inc

  NWENTRY_INFO = Record
    sequence : nuint32;
    parent : nuint32;
    attributes : nuint32;
    uniqueID : nuint8;
    flags : nuint8;
    nameSpace : nuint8;
    nameLength : nuint8;
    name : Array[0..11] Of nuint8;
    creationDateAndTime : nuint32;
    ownerID : nuint32;
    lastArchiveDateAndTime : nuint32;
    lastArchiverID : nuint32;
    dir : NWDIR_INFO
  End;
```

Fields

sequence

Specifies the sequence for iteratively scanning entries (-1 initially).

parent

Specifies the directory handle to parent directory.

attributes

Specifies the entry attributes.

uniqueID

Specifies the unique entry ID.

flags

Is reserved.

nameSpace

Specifies the name space creating the entry.

nameLength

Specifies the length of the *name* field.

name

Specifies the entry name.

creationDateAndTime

Specifies when the entry was created.

ownerID

Specifies the object ID of the owner.

lastArchiveDateAndTime

Specifies when the entry was last archived.

lastArchiverID

Specifies the ID of the object last archiving the entry.

Remarks

The *attributes* field can have the following values:

C Value	Pascal Value	Value Name
0x000000 00L	$000000 00	A_NORMAL
0x000000 01L	$000000 01	A_READ_ONLY
0x000000 02L	$000000 02	A_HIDDEN
0x000000	$000000	A_SYSTEM

04L	04	
0x00000008L	$00000008	A_EXECUTE_ONLY
0x00000010L	$00000010	A_DIRECTORY
0x00000020L	$00000020	A_NEEDS_ARCHIVED
0x00000080L	$00000080	A_SHAREABLE
0x00001000L	$00001000	A_TRANSACTIONAL
0x00002000L	$00002000	A_INDEXED
0x00004000L	$00004000	A_READ_AUDIT
0x00008000L	$00008000	A_WRITE_AUDIT
0x00010000L	$00010000	A_IMMEDIATE_PURGE
0x00020000L	$00020000	A_RENAME_INHIBIT
0x00040000L	$00040000	A_DELETE_INHIBIT
0x00080000L	$00080000	A_COPY_INHIBIT
0x00400000L	$00400000	A_FILE_MIGRATED
0x00800000L	$00800000	A_DONT_MIGRATE
0x02000000L	$02000000	A_IMMEDIATE_COMPRESS
0x04000000L	$04000000	A_FILE_COMPRESSED
0x08000000L	$08000000	A_DONT_COMPRESS
0x20000000L	$20000000	A_CANT_COMPRESS

The *nameSpace* field can have the following values:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM

3 NW_NS_FTAM
4 NW_NS_OS2
4 NW_NS_LONG

NWET_INFO

Returns directory entry trustee information
Service: File System
Defined In: nwdentry.h

Structure

```
typedef struct {
   nstr8          entryName[16];
   nuint32        creationDateAndTime;
   nuint32        ownerID;
   nuint32        sequenceNumber;
   TRUSTEE_INFO   trusteeList[20];
} NWET_INFO;
```

Pascal Structure

```
Defined in nwdentry.inc

NWET_INFO = Record
    entryName : Array[0..15] Of nstr8;
    creationDateAndTime : nuint32;
    ownerID : nuint32;
    sequenceNumber : nuint32;
    trusteeList : Array[0..19] Of TRUSTEE_INFO
  End;
```

Fields

entryName

Specifies the name of the entry (directory or file).

creationDateAndTime

Specifies the date and time when the entry was created.

ownerID

Specifies the ID of the owner for the entry.

sequenceNumber

Specifies the sequence for iteratively scanning entries.

trusteeList

Specifies an array of up to 20 TRUSTEE_INFO structures.

NWFILE_INFO

Defines entry information for files
Service: File System
Defined In: nwdentry.h

Structure

```
typedef struct {
   nuint32   updateDateAndTime;
   nuint32   updatorID;
   nuint32   fileSize;
   nuint8    reserved[44];
   nuint16   inheritedRightsMask;
   nuint16   lastAccessDate;
   nuint8    reserved2[28];
} NWFILE_INFO;
```

Pascal Structure

```
Defined in nwdentry.inc

  NWFILE_INFO = Record
    updateDateAndTime : nuint32;
    updatorID : nuint32;
    fileSize : nuint32;
    reserved : Array[0..43] Of nuint8;
    inheritedRightsMask : nuint16;
    lastAccessDate : nuint16;
    reserved2 : Array[0..27] Of nuint8
  End;
```

Fields

updateDateAndTime

Specifies when the file was last updated.

updatorID

Specifies the ID of the object that last updated the file.

fileSize

Specifies the size of the file.

reserved

Is reserved.

inheritedRightsMask

Specifies the Inherited Rights Mask for the file.

lastAccessDate

Specifies when the file was last accessed

reserved2

Is reserved.

Remarks

The *inheritedRightsMask* field can have the following values:

C Value	Pascal Value	Value Description
0x0000	$0000	TR_NONE
0x0001	$0001	TR_READ
0x0002	$0002	TR_WRITE
0x0004	$0004	TR_OPEN
0x0004	$0004	TR_DIRECTORY
0x0008	$0008	TR_CREATE
0x0010	$0010	TR_DELETE
0x0010	$0010	TR_ERASE
0x0020	$0020	TR_OWNERSHIP
0x0020	$0020	TR_ACCESS_CTRL
0x0040	$0040	TR_FILE_SCAN
0x0040	$0040	TR_SEARCH
0x0040	$0040	TR_FILE_ACCESS
0x0080	$0080	TR_MODIFY
0x01FB	$01FB	TR_ALL

0x0100	$0100	TR_SUPERVISOR
0x00FB	$00FB	TR_NORMAL

OPEN_FILE_CONN

Returns information about the open files for a connection
Service: File System
Defined In: nwfile.h

Structure

```
typedef struct {
   nuint16   taskNumber;
   nuint8    lockType;
   nuint8    accessControl;
   nuint8    lockFlag;
   nuint8    volNumber;
   nuint32   parent;
   nuint32   dirEntry;
   nuint8    forkCount;
   nuint8    nameSpace;
   nuint8    nameLen;
   nstr8     fileName[255];
} OPEN_FILE_CONN;
```

Pascal Structure

```
Defined in nwfile.inc

  OPEN_FILE_CONN = Record
    taskNumber : nuint16;
    lockType : nuint8;
    accessControl : nuint8;
    lockFlag : nuint8;
    volNumber : nuint8;
    parent : nuint32;
    dirEntry : nuint32;
    forkCount : nuint8;
    nameSpace : nuint8;
    nameLen : nuint8;
    fileName : Array[0..254] Of nstr8
  End;
```

Fields

taskNumber

Specifies the number of the task which has this file opened (each file can have multiple task numbers).

lockType

Specifies how the file is locked.

accessControl

Specifies how the file is being accessed.

lockFlag

Specifies whether the file is locked.

volNumber

Specifies the volume number (SYS is always 0).

parent

Specifies the ID number for the parent directory.

dirEntry

Specifies the directory entry number.

forkCount

Specifies the number of forks associated with the file.

nameSpace

Specifies the name space creating the file.

nameLen

Specifies the number of bytes in the filename.

fileName

Specifies the name of file (long names are supported).

Remarks

The first four fields contain information similar to their counterparts in the CONN_USING_FILE structure. The remaining fields identify the file and its name space.

The *lockType* field can have the following values:

0x01 Locked
0x02 Open shareable
0x04 Logged
0x08 Open Normal
0x40 TTS holding
0x80 Transaction flag set

The *accessControl* field can have the following values:

0x01 Open for read by this client
0x02 Open for write by this client
0x04 Deny read requests from others
0x08 Deny write requests from others
0x10 File detached
0x20 TTS holding detach
0x40 TTS holding open

The *lockFlag* field can have the following values:

0x00 Not locked
0xFE Locked by a file lock
0xFF Locked by begin share file set

The *nameSpace* field can have the following values:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_OS2
4 NW_NS_LONG

OPEN_FILE_CONN_CTRL

Returns a list of files a specified connection has open
Service: File System
Defined In: nwfile.h

Structure

```
typedef struct {
   nuint16   nextRequest;
   nuint16   openCount;
   nuint8    buffer[512];
   nuint16   curRecord;
} OPEN_FILE_CONN_CTRL;
```

Pascal Structure

```
Defined in nwfile.inc

OPEN_FILE_CONN_CTRL = Record
    nextRequest : nuint16;
    openCount : nuint16;
    buffer : Array[0..511] Of nuint8;
    curRecord : nuint16
  End;
```

Fields

nextRequest

Specifies an iterator.

openCount

Specifies the number of OPEN_FILE_CONN structures contained in the *buffer* field.

buffer

Specifies the returnedOPEN_FILE_CONN structure.

curRecord

Specifies the offset in the *buffer* field of the next record to return and is used internally by the **NWScanOpenFilesByConn2** function to track the next record to return in the OPEN_FILE_CONN structure.

PARAMETER_TABLE_TYPE

Service: File System
Defined In: nwconfig.h

Structure

```
typedef union {
   char N_FAR     *string;
   unsigned int    number;
   unsigned long   longNumber;
} PARMETER_TABLE_TYPE;
```

Fields

string

number

longNumber

SEARCH_DIR_INFO

Service: File System
Defined In: nwfile.h

Structure

```
typedef struct {
   nuint16   sequenceNumber;
   nuint16   reserved1;
   nstr8     directoryName[15];
   nuint8    directoryAttributes;
   nuint8    directoryAccessRights;
   nuint16   createDate;
   nuint16   createTime;
   nuint32   owningObjectID:
   nuint16   reserved2;
   nuint16   directoryStamp;
} SEARCH_DIR_INFO;
```

Pascal Structure

```
Defined in nwfile.inc

SEARCH_DIR_INFO = Record
    sequenceNumber : nuint16;
    reserved1 : nuint16;
    directoryName : Array[0..14] Of nstr8;
    directoryAttributes : nuint8;
    directoryAccessRights : nuint8;
    createDate : nuint16;
    createTime : nuint16;
    owningObjectID : nuint32;
    reserved2 : nuint16;
    directoryStamp : nuint16
  End;
```

Fields

sequenceNumber

Is reserved.

reserved1

Is reserved.

directoryName

Specifies the short name of the directory.

directoryAttributes

Specifies the attributes for the directory.

directoryAccessRights

Specifies the access rights.

createDate

Specifies the time the directory was created.

createTime

Specifies the date the directory was created.

owningObjectID

Specifies the ID of the object owning the directory.

reserved2

Is reserved.

directoryStamp

Specifies 0xD1D1 when returned.

Remarks

The *directoryAttributes* field can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x10	$10	FA_DIRECTORY

FA_DIRECTORY will always be in the bit mask for a directory.

The *directoryAccessRights* field can have the following values:

C Value	Pascal Value	Value Name
0x00	$00	TA_NONE
0x01	$01	TA_READ
0x02	$02	TA_WRITE
0x04	$04	TA_OPEN Obsolete in 3.x and above.

0x08	$08	TA_CREATE
0x10	$10	TA_DELETE
0x20	$20	TA_OWNERSHIP
0x40	$40	TA_SEARCH
0x80	$80	TA_MODIFY
0xFB	$FB	TA_ALL

SEARCH_FILE_INFO

Service: File System
Defined In: nwfile.h

Structure

```
typedef struct {
   nuint16   sequenceNumber;
   nuint16   reserved;
   nstr8     fileName[15];
   nuint8    fileAttributes;
   nuint8    fileMode;
   nuint32   fileLength;
   nuint16   createDate;
   nuint16   accessDate;
   nuint16   updateDate;
   nuint16   updateTime;
} SEARCH_FILE_INFO;
```

Pascal Structure

```
SEARCH_FILE_INFO = Record
    sequenceNumber : nuint16;
    reserved : nuint16;
    fileName : Array[0..14] Of nstr8;
    fileAttributes : nuint8;
    fileMode : nuint8;
    fileLength : nuint32;
    createDate : nuint16;
    accessDate : nuint16;
    updateDate : nuint16;
    updateTime : nuint16
  End;
```

Fields

sequenceNumber

Is reserved.

reserved

Is reserved.

fileName

Specifies the short name of the file.

fileAttributes

Specifies the attributes for the file.

fileMode

Specifies the access rights.

fileLength

Specifies the size of the file in bytes.

createDate

Specifies the date when the file was created.

accessDate

Specifies the date when the file was last accessed.

updateDate

Specifies the date when the file was last modified.

updateTime

Specifies the time when the file was last modified.

Remarks

The *fileAttributes* field can have the following values (may be ORed):

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

The *fileMode* field can have the following values:

0x01 Open for read by this client
0x02 Open for write by this client
0x04 Deny read requests from others
0x08 Deny write requests from others
0x10 File detached
0x20 TTS holding detach
0x40 TTS holding open

SetTableStruct

Service: File System
Defined In: nwconfig.h

Structure

```
typedef struct {
   int                 numberOfElements;
   int                *elementCode;
   char N_FAR *N_FAR  *elementName;
   int N_FAR          *elementValue;
} SetTableStruct;
```

Fields

numberOfElements

Specifies the number of set elements for which the parser should look.

elementCode

Is reserved.

elementName

Points to a character array containing the string for which to look.

elementValue

Points to an array of integers.

Remarks

The corresponding entry which is matched by the *elementName* field will be used as the value for the *elementValue* field.

stat (Structure)

Holds information about the status of a file or directory
Service: File System
Defined In: sys\stat.h

Structure

```
struct stat {
   dev_t            st_dev;
   ino_t            st_ino;
   unsigned short   st_mode;
   short            st_nlink;
   unsigned long    st_uid;
   short            st_gid;
   dev_t            st_rdev;
   off_t            st_size;
   time_t           st_atime;
   time_t           st_mtime;
   time_t           st_ctime;
   time_t           st_btime;
   unsigned long    st_attr;
   unsigned long    st_archievedID;
   unsigned long    st_updatedID;
   unsigned short   st_inheritedRightsMask;
   unsigned char    st_originatingNameSpace;
   unsigned char    st_name[255+1];
   size_t           st_blksize;
   size_t           st_blocks;
   unsigned int     st_flags;
   unsigned long    st_spare[4];
};
```

Fields

st_dev
Specifies the volume number.

st_ino
Specifies the directory entry of the st_name.

st_mode
Specifies the emulated file mode.

st_nlink
Specifies the count of hard links (always 1).

d_ino
Specifies the object ID of the owner.

st_gid

Specifies the group ID (always 0).

st_rdev

Specifies the device type (always 0).

off_t

Specifies the total file size (files only).

st_atime

Specifies the last access date (files only) in calendar time (seconds since the Jan.1, 1970 (UTC)).

st_mtime

Specifies the last modify date and time in calendar time.

st_ctime

POSIX: Specifies the last status change time in calendar time.

st_attr

Specifies the file attribute as defined in NWFATTR.H.

st_archivedID

Specifies the user/object ID of the last archive.

st_updatedID

Specifies the user/object ID of the last update.

st_inheritiedRightsMask

Specifies the NDS inherited rights mask.

st_originatingNameSpace

Specifies the namespace of creation.

st_name

Specifies the target namespace.

st_blksize

Specifies the block size for allocation (files only).

st_blocks

Specifies the count of blocks allocated to the file.

st_flags

Specifies user-defined flags.

st_name

Is reserved.

TRUSTEE_INFO

Contains a directory trustee with the object rights
Service: File System
Defined In: nwdirect.h

Structure

```
typedef struct {
   nuint32   objectID;
   nuint16   objectRights;
} TRUSTEE_INFO;
```

Pascal Structure

```
 TRUSTEE_INFO = Record
    objectID : nuint32;
    objectRights : nuint16
  End;
```

Fields

objectID

Specifies the ID of the object.

objectRights

Specifies the rights the object has on a directory.

TypeDefaultStruct

Service: File System
Defined In: nwconfig.h

Structure

```
typedef struct {
   int    paramType;
   long   defaultValue;
} TypeDefaultStruct;
```

Fields

paramType

defaultValue

VOLUME_STATS

Holds volume information
Service: File System
Defined In: nwdir.h

Structure

```
typedef struct tagVOLUME_STATS {
   long   systemElapsedTime;
   BYTE   volumeNumber;
   BYTE   logicalDriveNumber;
   WORD   sectorsPerBlock;
   long   startingBlock;
   WORD   totalBlocks;
   WORD   availableBlocks;
   WORD   totalDirectorySlots;
   WORD   availableDirecotrySlots;
   WORD   maxDirectorySlotsUsed;
   BYTE   isHashing;
   BYTE   isRemovable;
   BYTE   isMounted;
   char   volumeName[17];
   LONG   purgableBlocks;
   LONG   notyetPurgableBlocks;
} VOLUME_STATS;
```

Defined In

nwdir.h

Fields

systemElapsedTime

Specifies the time in seconds since the system was brought up.

volumeNumber

Specifies the volume number (same as the Volume Table number for the server).

logicalDriveNumber

Specifies the logical drive number.

sectorsPerBlock

Specifies the number of 512-byte sectors in a block for the volume.

startingBlock

Specifies the starting block of the volume.

totalBlocks

Specifies the total number of blocks in the volume.

availableBlocks

Specifies the number of available blocks on the volume.

totalDirectorySlots

Specifies the total number of directory slots on the volume.

availableDirectorySlots

Specifies the number of available directory slots on the volume.

maxDirectorySlotsUsed

Specifies the maximum number of directory slots used on the volume.

isHashing

Specifies whether the volume is hashing.

isRemovable

Specifies whether the volume is removable (always non-zero for NetWare 3.x and 4.x):

non-zero Volume can be removed
0x00 Volume cannot be removed

isMounted

Specifies whether the volume is mounted.

volumeName

Specifies the volume name (2-15 characters plus the NULL terminator).

purgableBlocks

Specifies the number of purgeable blocks

notYetPurgableBlocks

Specifies the number of blocks not yet purgeable.

Remarks

The *volumeName* field cannot contain spaces or the following characters:

*	Asterisk
?	Question mark
:	Colon
	Slash
	Backslash

VOLUME_INFO

Contains volume information
Service: File System
Defined In: nwdir.h

Structure

```
typedef struct tagVOLUME_INFO {
   long   systemElapsedTime;
   BYTE   volumeNumber;
   BYTE   logicalDriveNumber;
   WORD   sectorsPerBlock;
   short  startingBlock;
   LONG   totalBlocks;
   LONG   availableBlocks;
   LONG   totalDirectorySlots;
   LONG   availableDirecotrySlots;
   BYTE   isHashing;
   BYTE   isRemovable;
   BYTE   isMounted;
   char   volumeName[17];
   LONG   purgableBlocks;
   LONG   notyetPurgableBlocks;
} VOLUME_INFO;
```

Fields

systemElapsedTime

Specifies the time in seconds since the system was brought up.

volumeNumber

Specifies the volume number (same as the Volume Table number).

logicalDriveNumber

Specifies the logical drive number.

sectorsPerBlock

Specifies the number of 512-byte sectors in a block for the volume.

startingBlock

Specifies the starting block of the volume.

totalBlocks

Specifies the total number of blocks in the volume.

availableBlocks

Specifies the number of available blocks on the volume.

totalDirectorySlots

Specifies the total number of directory slots on the volume.

availableDirectorySlots

Specifies the number of available directory slots on the volume.

isHashing

Specifies whether the volume is hashing.

isRemovable

Specifies whether the volume is removable (always non-zero for NetWare 3.x and 4.x):

non-zero Volume can be removed

0x00 Volume cannot be removed

isMounted

Specifies whether the volume is mounted.

volumeName

Specifies the volume name (2-15 characters plus the NULL terminator).

purgableBlocks

Specifies the number of purgeable blocks

notYetPurgableBlocks

Specifies the number of blocks not yet purgeable.

Remarks

The *volumeName* field cannot contain spaces or the following characters:

*	Asterisk
?	Question mark
:	Colon
	Slash
	Backslash

File System Monitoring

File System Monitoring: Guides

File System Monitoring: General Guide

File System Monitoring Introduction

Registering for Callback

Potential Uses

Writing a File System Monitor NLM

File System Monitoring Functions

Additional Links

File System Monitoring: Functions

File System Monitoring: Structures

Parent Topic:

File Overview

File System Monitoring: Task Guide

Writing a File System Monitor NLM

Additional Links

File System Monitoring: Functions

File System Monitoring: Structures

Parent Topic:

File System Monitoring: Guides

File System Monitoring: Concept Guide

File System Monitoring Introduction

Registering for Callback

Pre-Execution and Post-Execution Monitoring

Additional Links

Parent Topic:

Registering for Callback

The NetWare® OS transfers control to yourNLM™ whenever it receives a request from any of its clients for a function that you have registered for monitoring. Control is transferred to a function in your NLM that has restrictions imposed on it by the NetWare OS.

This "callback function" is required to have parameters that the OS is expecting and can fill out. Your NLM or a system administrator can then use the information passed to the callback function by the OS to decide what action to take, if any, before or after the request is filled.

It's as if, when you call **NWAddFSMonitorHook**, your NLM is given a window through which the NetWare OS looks at every request for a file system function that you have registered for monitoring. Your NLM can then test each one against a selected set of conditions, such as the presence of a virus. In the event your NLM detects something suspicious, it can alter or fail the request or make a record of it for the system administrator to act upon later.

Parent Topic:

File System Monitoring: Guides

The Monitoring Function

What your monitoring function returns depends on whether it is a pre-execution callback or a post-execution callback (see "Pre-Execution and Post-Execution Monitoring").

Pre-Execution Callbacks

Post-Execution Callbacks

Parent Topic:

Registering for Callback

Potential Uses

Novell® originally created File System Monitoring to fill a demand for a virus detection/protection hook. Because viruses can infect mission-critical files, this is a vitally important use of the service, but not the only one. File System Monitoring could also be used for any other network service that relies on monitoring file system requests. A couple of these are hot backup and version control.

Hot Backup

Version Control

Parent Topic:

File System Monitoring: Guides

File System Monitoring: Tasks

Writing a File System Monitor NLM

The four steps below are the essential parts of writing a file system monitor NLM™. They are taken from the example NLM, FSHOOK.C, in the EXAMPLES directory.

1. **Create your callback functions.**

```
int openFileCallBackFunc(OpenFileCallBackStruct *ofcbs)
   static int cnt = 0;
   char   user[48];
   int    ccode;
   WORD   objType;
   long   objID;
   BYTE   loginTime[7];
   LONG   pc;
   BYTE   ps[255];
   BYTE   volName[16];
   LONG   prevThreadGroupID;
   prevThreadGroupID = SetThreadGroupID(mainThreadGroupID);
   ccode = GetConnectionInformation(ofcbs->connection, user,
                           &objType,&objID, loginTime);
   if (ccode != 0)
      return 0xFF;
   printf("%dth  OPEN request.  by %s (connNum %d), ", ++cnt, user,
            ofcbs->connection);
   FEMapVolumeNumberToName(ofcbs->volume, volName);
   for (pc = 1; pc <= volName[0]; pc++)
      putchar(volName[pc]);
   putchar(':');
   FEMapVolumeAndDirectoryToPath(ofcbs->volume,
                        ofcbs->dirBase, ps, &pc);
   if (ps[0])
      printNetWareStr(pc, ps);
   printNetWareStr(ofcbs->pathComponentCount, ofcbs->pathString);
   putchar('\n');
   SetThreadGroupID(prevThreadGroupID);
   return 0;
}
```

Registering your callback functions tells the OS to transfer control to your NLM whenever a specified file system event is triggered. These callback functions can be thought of as "windows" to the file system, which are opened by calling **NWAddFSMonitorHook**.

which are opened by calling **NWAddFSMonitorHook**.

The example callback function above, **openFileCallBackFunc**, receives the OpenFileCallBackStruct pointer and prints out some of its field values for informational purposes. These structures are defined in the NWFSHOOK.H file. For example, the structure returned when the OS opens a file (defined in NWFSHOOK.H) is shown below:

```
typedef struct {
   LONG      connection;
   LONG      task;
   LONG      volume;
   LONG      dirBase;
   BYTE      *pathString;
   LONG      pathComponentCount;
   LONG      nameSpace;
   LONG      attributeMatchbits;
   LONG      requestedAccessRights;
   LONG      dataStreamNumber;
} OpenFileCallBackStruct;
```

2. **Begin monitoring.**

```
ccode = NWAddFSMonitorHook(FSHOOK_PRE_OPENFILE,
    openFileCallBackFunc, &preOpenFileHandle);
if (ccode != 0)
{
   printf("nwaddfsmonitorhook error.  ccode: %#x, hook:  openFile
           ccode);
}
```

By calling **NWAddFSMonitorHook**, register your callback functions and start monitoring. In this case, the **openFileCallBackFunc** has been registered to be called back after the OS executes the file opening function (by specifying FSHOOK_POST_OPENFILE).

3. **Wait for callbacks from the OS.**

```
while (1)
   ThreadSwitchWithDelay(1000);     //sleep forever.....until unload
```

Provide a mechanism, like sleeping forever (above), for keeping the NLM inactive but loaded and ready to respond to a callback from the OS.

4. **Stop monitoring.**

```
void ExitandRemoveMonitorHooks()
{
    NWRemoveFSMonitorHook(FSHOOK_PRE_OPENFILE,   openFileCallBackFun
}
```

Deregister the callback by calling **NWRemoveFSMonitorHook**.

Parent Topic:

File System Monitoring: Guides

File System Monitoring: Concepts

Callback Structures

The following table summarizes the structures returned by file system monitoring callbacks. Descriptions of all structures can be found in File System Monitoring: Structures.

Table auto. File System Monitoring Callback Structures

Callback (before and after OS Execution)	**Structure returned by Callback**
FSHOOK_PRE_ERASEFILE FSHOOK_POST_ERASEFILE	EraseFileCallBackStruct
FSHOOK_PRE_OPENFILE FSHOOK_POST_OPENFILE	OpenFileCallBackStruct
FSHOOK_PRE_CREATEFILE FSHOOK_POST_CREATEFILE	CreateFileCallBackStruct
FSHOOK_PRE_CREATE_OPENFILE FSHOOK_POST_CREATE_OPENFILE	CreateAndOpenCallBackStruct
FSHOOK_PRE_RENAME_OR_MOVE FSHOOK_POST_RENAME_OR_MOVE	RenameMoveEntryCallBackStruct
FSHOOK_PRE_CLOSEFILE FSHOOK_POST_CLOSEFILE	CloseFileCallBackStruct
FSHOOK_PRE_CREATEDIR FSHOOK_POST_CREATEDIR	CreateDirCallBackStruct
FSHOOK_PRE_DELETEDIR FSHOOK_POST_DELETEDIR	DeleteDirCallBackStruct
FSHOOK_PRE_MODIFY_DIRENTRY FSHOOK_POST_MODIFY_DIRENTRY	ModifyDirEntryCallBackStruct
FSHOOK_PRE_SALVAGE_DELETED FSHOOK_POST_SALVAGE_DELETED	SalvageDeletedCallBackStruct
FSHOOK_PRE_PURGE_DELETED FSHOOK_POST_PURGE_DELETED	PurgeDeletedCallBackStruct

FSHOOK_PRE_RENAME_NS_ENTRY FSHOOK_POST_RENAME_NS_ENTRY	RenameNSEntryCallBackStruct
FSHOOK_PRE_GEN_SALVAGE_DELETED FSHOOK_POST_GEN_SALVAGE_DELETED	GenericSalvageDeletedCBStruct
FSHOOK_PRE_GEN_PURGE_DELETED FSHOOK_POST_GEN_PURGE_DELETED	GenericPurgeDeletedCBStruct
FSHOOK_PRE_GEN_OPEN_CREATE FSHOOK_POST_GEN_OPEN_CREATE	GenericOpenCreateCBStruct
FSHOOK_PRE_GEN_RENAME FSHOOK_POST_GEN_RENAME	GenericRenameCBStruct
FSHOOK_PRE_GEN_ERASEFILE FSHOOK_POST_GEN_ERASEFILE	GenericEraseFileCBStruct
FSHOOK_PRE_GEN_MODIFY_DOS_INFO FSHOOK_POST_GEN_MODIFY_DOS_INFO	GenericModifyDOSInfoCBStruct
FSHOOK_PRE_GEN_MODIFY_NS_INFO FSHOOK_POST_GEN_MODIFY_NS_INFO	GenericModifyNSInfoCBStruct

Parent Topic:

Registering for Callback

Related Topics:

Pre-Execution and Post-Execution Monitoring

The Monitoring Function

File System Monitoring Functions

Function	Purpose
NWAddFSMonitorHook	Begin monitoring the file system
NWRemoveFSMonitorHook	Stop monitoring the file system

Parent Topic:

File System Monitoring: Guides

File System Monitoring Introduction

File System Monitoring allows your NLM™ application to "hook" the file system functions that correspond to the list below. Before any of these functions that your NLM has registered for callback are executed by the NetWare® OS, your NLM has the option of changing it, failing it, or simply making a record of its execution.

file erasing

file opening

file creating

file creating/opening

file renaming/moving

file closing

directory creating

directory deleting

directory entry modification

salvaging

purging

name space entry renaming

generic salvaging

generic purging

generic opening/creating

generic renaming

generic file erasing

generic DOS information modification

generic name space information modification

Parent Topic:

Hot Backup

A hot backup NLM™ could register functions that create and modify files, putting the results in a special log file. Then, from time to time, it could back up all the new material to a specified medium. This would eliminate the need for humanly-executed backup.

Parent Topic:

Potential Uses

Related Topics:

Version Control

Post-Execution Callbacks

If you are registering a post-execution function, it should return 2 parameters, a pointer to the structure returned for the OS function and a completion code indicating whether or not the OS function completed successfully.

> **NOTE:** The post-execution callback function must not sleep, because the fields in the return structure are subject to change.

Parent Topic:

The Monitoring Function

Related Topics:

Pre-Execution Callbacks

Pre-Execution and Post-Execution Monitoring

When registering a callback function, you specify in the *callBackNumber* parameter whether the callback is made before or after the OS executes the function. Possible values for the *callBackNumber* include both a "pre" and "post" version for every OS function that can be monitored. The "pre" versions callback to your function before the OS function executes, whereas the "post" versions callback to your function after the OS function executes. If the callback occurs before the OS executes the function, your NLM™ can fail that function. Call **NWAddFSMonitorHook** once for each function you want to be monitored.

The name space entry changing hooks and all generic hooks are used for monitoring functions called from other than DOS clients. These non-DOS

hooks are supported only on NetWare® versions 3.12 and higher, while the remaining hooks are also supported on version 3.11. The following table lists the values for *callBackNumber* for each OS function.

Table auto. Values for callBackNumber

OS Function to Monitor	**Callback before OS Execution**	**Callback after OS Execution**
file erasing	FSHOOK_PRE_ERASEFILE	FSHOOK_POST_ERASEFILE
file opening	FSHOOK_PRE_OPENFILE	FSHOOK_POST_OPENFILE
file creating	FSHOOK_PRE_CREATEFILE	FSHOOK_POST_CREATEFILE
file creating/ opening	FSHOOK_PRE_CREATE_OPENFILE	FSHOOK_POST_CREATE_OPENFILE
file renaming /moving	FSHOOK_PRE_RENAME_OR_MOVE	FSHOOK_POST_RENAME_OR_MOVE
file closing	FSHOOK_PRE_CLOSEFILE	FSHOOK_POST_CLOSEFILE
directory creating	FSHOOK_PRE_CREATEDIR	FSHOOK_POST_CREATEDIR
directory deleting	FSHOOK_PRE_DELETEDIR	FSHOOK_POST_DELETEDIR
directory entry modification	FSHOOK_PRE_MODIFY_DIRENTRY	FSHOOK_POST_MODIFY_DIRENTRY
salvaging	FSHOOK_PRE_SALVAGE_DELETED	FSHOOK_POST_SALVAGE_DELETED
purging	FSHOOK_PRE_PURGE_DELETED	FSHOOK_POST_PURGE_DELETED
name space entry renaming	FSHOOK_PRE_RENAME_NS_ENTRY	FSHOOK_POST_RENAME_NS_ENTRY
generic salvaging	FSHOOK_PRE_GEN_SALVAGE_DELETED	FSHOOK_POST_GEN_SALVAGE_DELETED
generic purging	FSHOOK_PRE_GEN_PURGE_DELETED	FSHOOK_POST_GEN_PURGE_DELETED
generic opening/	FSHOOK_PRE_GEN_OPEN_CREATE	FSHOOK_POST_GEN_OPEN_CREATE

creating		
generic renaming	FSHOOK_PRE_GEN_RENAME	FSHOOK_POST_GEN_RENAME
generic file erasing	FSHOOK_PRE_GEN_ERASEFILE	FSHOOK_POST_GEN_ERASEFILE
generic DOS information modification	FSHOOK_PRE_GEN_MODIFY_ DOS_INFO	FSHOOK_POST_GEN_MODIFY_DOS_INFO
generic name space information modification	FSHOOK_PRE_GEN_MODIFY_NS_INFO	FSHOOK_POST_GEN_MODIFY_NS_INFO

Parent Topic:

Registering for Callback

Pre-Execution Callbacks

If you are registering a pre-execution function, it should return one parameter, a pointer to the structure returned for the OS function you are monitoring.

In the case of pre-execution callbacks, you have the option of failing the OS function and returning an error. If your NLM™ decides to fail a request, it should return one of the OS's standard error codes (see NITERROR.H).

Parent Topic:

The Monitoring Function

Related Topics:

Post-Execution Callbacks

Version Control

Likewise, a version control NLM™ could keep a record of .obj files that have been created or modified and store a copy of the last one, along with all pertinent information, in a specified place.

Parent Topic:

Potential Uses

Related Topics:

Hot Backup

File System Monitoring: Functions

NWAddFSMonitorHook

Allows the application to monitor ("hook") various OS file system routines
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: File System Monitoring

Syntax

```
#include <nwfshook.h>

LONG NWAddFSMonitorHook (
   LONG   callBackNumber,
   void   *callBackFunc,
   LONG   *callBackHandle);
```

Parameters

callBackNumber

(IN) Specifies which type of OS file system routine you want to hook.

callBackFunc

(IN) Points to the function that you want the OS to call (pass control to) when the hooked file system routine is going to be or has been called by a client or any NLM™ application on the local server.

callBackHandle

(OUT) Receives a handle that identifies the file system monitor hook. This handle is passed to **NWRemoveFSMonitorHook** when removing the hook.

Return Values

This function returns 0 if the OS routine corresponding to the *callBackNumber* was successfully "hooked." NetWare® errors are returned if the OS routine was not hooked.

Remarks

The *callBackNumber* parameter specifies the OS file system routine that you want to hook, and whether the *callBackFunc* is called before (a "pre OS call hook") or after (a "post OS call hook") the OS routine executes. The last eight sets of hooks (FSHOOK_PRE/POST_RENAME_NS_ENTRY and all generics) are used for tracking routines called from other than DOS clients (note that non-DOS hooks are available for use only with

NetWare versions 3.12 and higher.) Hooks cannot be ORed, so you must call **NWAddFSMonitorHook** once for each routine you want to be monitored. Values for *callBackNumber* and the OS routines that they hook are defined in nwfshook.h and listed below:

OS Routine to Monitor	Callback before/after OS Execution	NetWare Version
file erasing	FSHOOK_PRE_ERASEFILE FSHOOK_POST_ERASEFILE	3.11 and higher
file opening	FSHOOK_PRE_OPENFILE FSHOOK_POST_OPENFILE	3.11 and higher
file creating	FSHOOK_PRE_CREATEFILE FSHOOK_POST_CREATEFILE	3.11 and higher
file creating/opening	FSHOOK_PRE_CREATE_OPENFILE FSHOOK_POST_CREATE_OPENFILE	3.11 and higher
file renaming/moving	FSHOOK_PRE_RENAME_OR_MOVE FSHOOK_POST_RENAME_OR_MOVE	3.11 and higher
file closing	FSHOOK_PRE_CLOSEFILE FSHOOK_POST_CLOSEFILE	3.11 and higher
directory creating	FSHOOK_PRE_CREATEDIR FSHOOK_POST_CREATEDIR	3.11 and higher
directory deleting	FSHOOK_PRE_DELETEDIR FSHOOK_POST_DELETEDIR	3.11 and higher
directory entry modification	FSHOOK_PRE_MODIFY_DIRENTRY FSHOOK_POST_MODIFY_DIRENTRY	3.11 and higher
salvaging	FSHOOK_PRE_SALVAGE_DELETED FSHOOK_POST_SALVAGE_DELETED	3.11 and higher
purging	FSHOOK_PRE_PURGE_DELETED FSHOOK_POST_PURGE_DELETED	3.11 and higher
name space entry renaming	FSHOOK_PRE_RENAME_NS_ENTRY FSHOOK_POST_RENAME_NS_ENTRY	3.12, 4.x
generic salvaging	FSHOOK_PRE_GEN_SALVAGE_DELETED FSHOOK_POST_GEN_SALVAGE_	3.12, 4.x

	DELETED	
generic purging	FSHOOK_PRE_GEN_PURGE_DELETED FSHOOK_POST_GEN_PURGE_DELETED	3.12, 4.x
generic opening/creating	FSHOOK_PRE_GEN_OPEN_CREATE FSHOOK_POST_GEN_OPEN_CREATE	3.12, 4.x
generic renaming	FSHOOK_PRE_GEN_RENAME FSHOOK_POST_GEN_RENAME	3.12, 4.x
generic file erasing	FSHOOK_PRE_GEN_ERASEFILE FSHOOK_POST_GEN_ERASEFILE	3.12, 4.x
generic DOS information modification	FSHOOK_PRE_GEN_MODIFY_DOS_INFO FSHOOK_POST_GEN_MODIFY_DOS_INFO	3.12, 4.x
generic name space information modification	FSHOOK_PRE_GEN_MODIFY_NS_INFO FSHOOK_POST_GEN_MODIFY_NS_INFO	3.12, 4.x

The *callBackFunc* parameter points to the callback function you have created. The number of parameters you should declare in *callBackFunc* varies depending on when the OS calls back the function.

The first parameter for both types of callback function is a pointer to the structure returned by the OS for the OS file system routine that is being monitored (for example, if you are monitoring file opens, the OS would return an OpenFileCallBackStruct). These callback structures are defined in nwfshook.h.

If you have specified a pre OS call back hook, this is the only parameter for the *callBackFunc*. If you have specified a post OS call back hook, the *callBackFunc* receives a second parameter, a pointer to a LONG value which is the completion code of the OS routine that you have hooked. The following illustrates what these functions would look like if you are monitoring file opens:

```
int PreCallBackFunc (OpenFileCallBackStruct const *structure);

void PostCallBackFunc (OpenFileCallBackStruct const *structure, LONG *c
```

Definitions of the structures returned by the callback function are described in File System Monitoring: Structures.

NOTE: If you specify a post OS call back hook, your callback function must not go to sleep, because the values in the callback structure can change before your thread wakes up again.

See Also

NWRemoveFSMonitorHook

NWRemoveFSMonitorHook

Removes a "hook" that is monitoring an OS file system routine
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: No
Service: File System Monitoring

Syntax

```
#include <nwfshook.h>

LONG NWRemoveFSMonitorHook (
   LONG   callBackNumber,
   LONG   callBackHandle);
```

Parameters

callBackNumber

(IN) Specifies the OS file system routine that you want to remove a hook from. See **NWAddFSMonitorHook** for possible values for this parameter.

callBackHandle

(IN) Specifies the handle that was returned when the hook was added by calling **NWAddFSMonitorHook**.

Return Values

This function returns 0 if the hook corresponding to*callBackNumber* was successfully removed. NetWare errors are returned if the OS routine was not successfully "unhooked."

Remarks

This function removes monitoring hooks from OS file system routines.

See Also

NWAddFSMonitorHook

File System Monitoring: Structures

CloseFileCallBackStruct

Contains information about a close file operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   fileHandle;
} CloseFileCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

fileHandle

Contains the NetWare file handle of the file.

CreateDirCallBackStruct

Contains information about a create directory operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   volume;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   pathComponentCount;
   LONG   nameSpace;
   LONG   directoryAccessMask;
} CreateDirCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

nameSpace

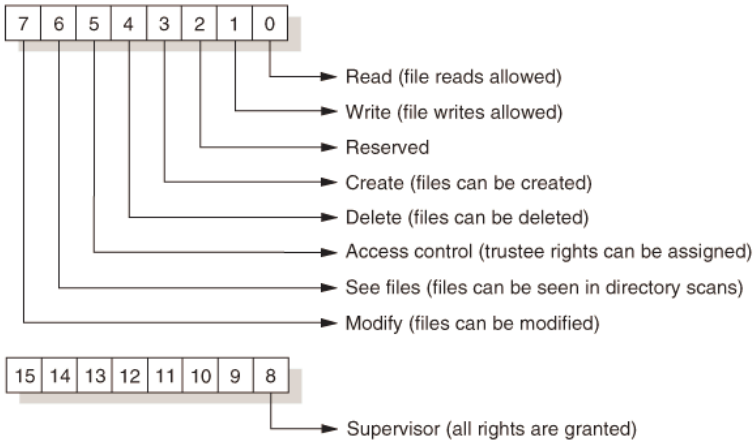
Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS

3	FTAM
4	OS2
5	NT

directoryAccessMask

Contains a bit mask by which the directory is to be accessed subsequently. This is the same bit mask used by ModifyInheritedRightsMask, shown in the following figure:



CreateFileCallBackStruct

Contains information about a create file operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   pathComponentCount;
   LONG   nameSpace;
   LONG   createAttributeBits;
   LONG   createFlagBits;
   LONG   dataStreamNumber;
} CreateFileCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

createAttributeBits

Contains the file attributes that the file is to have when it is created.

createFlagBits

Contains flags that can be set to allow more flexibility in the create operation. These bits are listed in the following table.

Bit	Meaning
DELETE_FILE_ON_CREATE_BIT	If the file already exists, it is deleted. This allows the file to be created again.
NO_RIGHTS_CHECK_ON_OPEN_BIT	The user's rights to the file are not checked when the file is opened.
NO_RIGHTS_CHECK_ON_CREATE_BIT	The user's rights to the file are not checked when the file is created.
FILE_WRITE_THROUGH_BIT	When a file write is performed, the write function does not return until the data is actually written to the disk.
ENABLE_IO_ON_COMPRESSED_DATA_BIT	Any subsequent I/O on this entry is compressed
LEAVE_FILE_COMPRESSED_DATA_BIT	After all I/O has been done, leave this file compressed

dataStreamNumber

Contains a number identifying the data stream type of the file or directory. The possible data streams are listed in the following table.

Number	Type
0	Primary Data Stream (corresponds to DOS)

1	Macintosh Resource Fork
2	FTAM Extra Data Fork

CreateAndOpenCallBackStruct

Contains information about a create/open operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   pathComponentCount;
   LONG   nameSpace;
   LONG   createAttributeBits;
   LONG   requestedAccessRights;
   LONG   createFlagBits;
   LONG   dataStreamNumber;
} CreateAndOpenCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

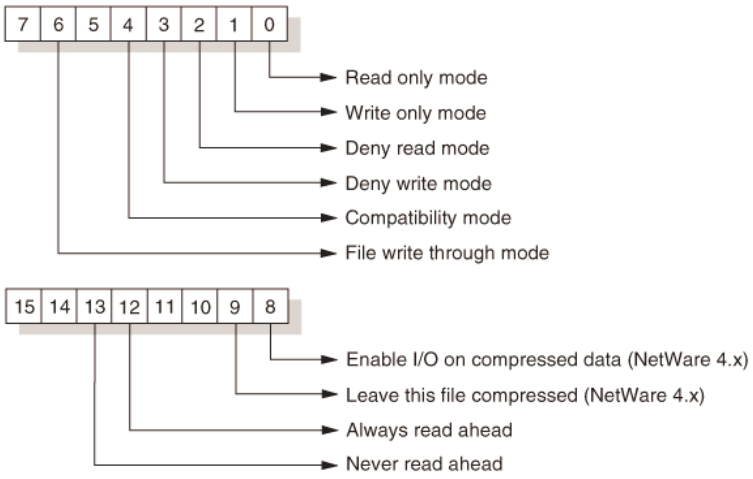
Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

createAttributeBits

Contains the file attributes that the file is to have when it is created.

requestedAccessRights

Indicates how the entry is to be opened, such as Read Only, Read Write, Compatibility mode, and so on. The bits in this mask are defined in the following figure.



createFlagBits

Contains flags that can be set to allow more flexibility in the create operation. These bits are listed in the following table.

Bit	Meaning
DELETE_FILE_ON_CREATE_BIT	If the file already exists, it is deleted. This allows the file to be created again.
NO_RIGHTS_CHECK_ON_OPEN_BIT	The user's rights to the file are not checked when the

	file is opened.
NO_RIGHTS_CHECK_ON_CREATE_BIT	The user's rights to the file are not checked when the file is created.
FILE_WRITE_THROUGH_BIT	When a file write is performed, the write function does not return until the data is actually written to the disk.
ENABLE_IO_ON_COMPRESSED_DATA_BIT	Any subsequent I/O on this entry is compressed
LEAVE_FILE_COMPRESSED_DATA_BIT	After all I/O has been done, leave this file compressed

dataStreamNumber

Contains a number identifying the data stream type of the file or directory. The possible data streams are listed in the following table.

Number	**Type**
0	Primary Data Stream (corresponds to DOS)
1	Macintosh Resource Fork
2	FTAM Extra Data Fork

DeleteDirCallBackStruct

Contains information about a delete directory operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   volume;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   pathComponentCount;
   LONG   nameSpace;
} DeleteDirCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM

4	OS2
5	NT

EraseFileCallBackStruct

Contains information about an erase file operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   pathComponentCount;
   LONG   nameSpace;
   LONG   attributeMatchBits;
} EraseFileCallBackStruct;
```

Defined In

nwfshook

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

nameSpace

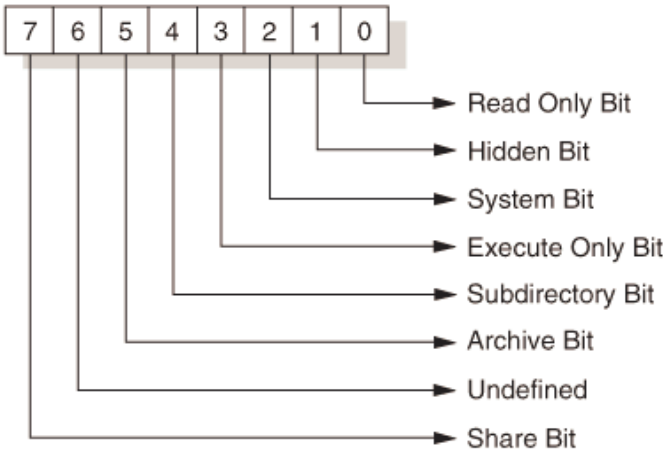
Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS

1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

attributeMatchBits

Contains a bit mask of the file attributes that are affected by this operation. That is, entries that have file attributes matching this bit mask are affected. The first byte of the file attributes mask is shown in the following figure. For more about the file attributes mask, see File Attributes.



GenericEraseFileCBStruct

Contains information about a generic erase file operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   pathComponentCount;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   nameSpace;
   LONG   searchAttributes;
} GenericEraseFileCBStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS

1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

searchAttributes

Contains a bit mask of the file attributes that are affected by this operation. That is, entries that have file attributes matching this bit mask are affected.

GenericModifyDOSInfoCBStruct

Contains information about a generic modify DOS information operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   pathComponentCount;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   nameSpace;
   LONG   searchAttributes;
   LONG   modifyMask;
   void   *modifyInfo;
   } GenericModifyDOSInfoCBStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

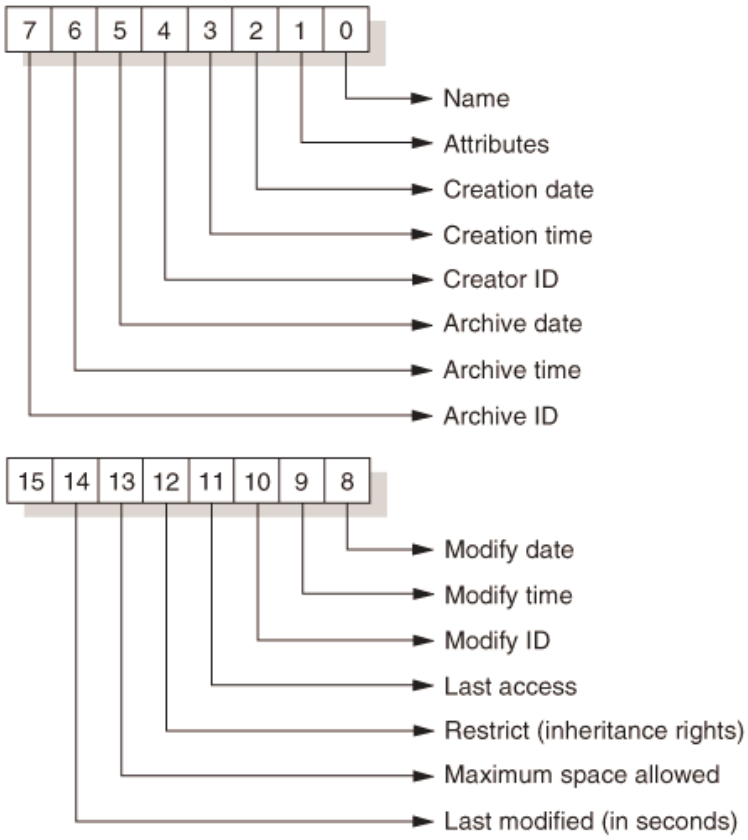
Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

searchAttributes

Contains field contains a bit mask of the file attributes that are affected by this operation. That is, entries that have file attributes matching this bit mask are affected.

modifyMask

Contains a bit mask that defines the items to be modified by this operation, see the following figure.



modifyInfo

Contains the data that is to replace the old data for this entry.

GenericModifyNSInfoCBStruct

Contains information about a generic modify name space information operation

Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   dataLength;
   LONG   srcNameSpace;
   LONG   dstNameSpace;
   LONG   volume;
   LONG   dirBase;
   LONG   modifyMask;
   void   *modifyInfo;
   } GenericModifyNSInfoCBStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

dataLength

Contains the size of the data in the *modifyInfo* field.

srcNameSpace

Contains the name space of the source. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2

5	NT

dstNameSpace

Contains the name space of the destination (see above).

volume

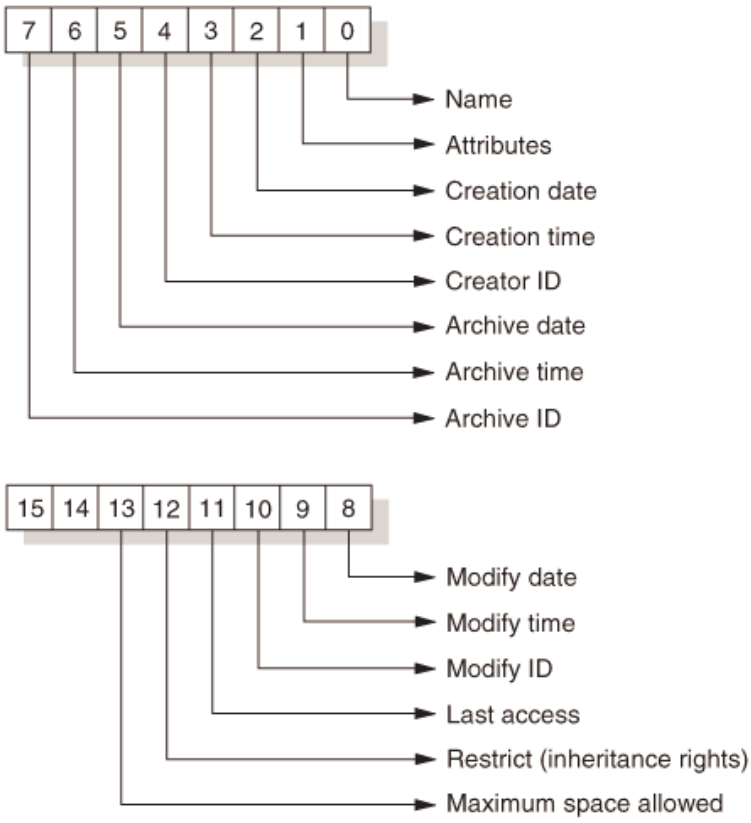
Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

modifyMask

Contains a bit mask that defines the items to be modified by this operation (see the following figure). Note that this bit mask differs slightly from the modify mask for the generic modify DOS information structure, in that it does not contain the "Last modified" bit.



modifyInfo

Contains the data that is to replace the old data for this entry.

GenericOpenCreateCBStruct

Contains information about a generic open/create operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   pathComponentCount;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   nameSpace;
   LONG   dataStreamNumber;
   LONG   openCreateFlags;
   LONG   searchAttributes;
   LONG   createAttributes;
   LONG   requestedAccessRights;
   LONG   returnInfoMask;
   LONG   *fileHandle;
   BYTE   *openCreateAction;
} GenericOpenCreateCBStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

pathComponentCount

Contains the number of components in the path.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

nameSpace

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

dataStreamNumber

Contains a number identifying the data stream type of the file or directory. The possible data streams are listed in the following table.

Number	Type
0	Primary Data Stream (corresponds to DOS)
1	Macintosh Resource Fork
2	FTAM Extra Data Fork

openCreateFlags

Contains the operation requested, such as opening a file, creating a file, and so on. The possible values are listed in the following table.

Value	Meaning
0x01	Open
0x02	Truncate
0x08	Create

searchAttributes

Contains a bit mask of the file attributes that are affected by this operation. That is, entries that have file attributes matching this bit mask are affected.

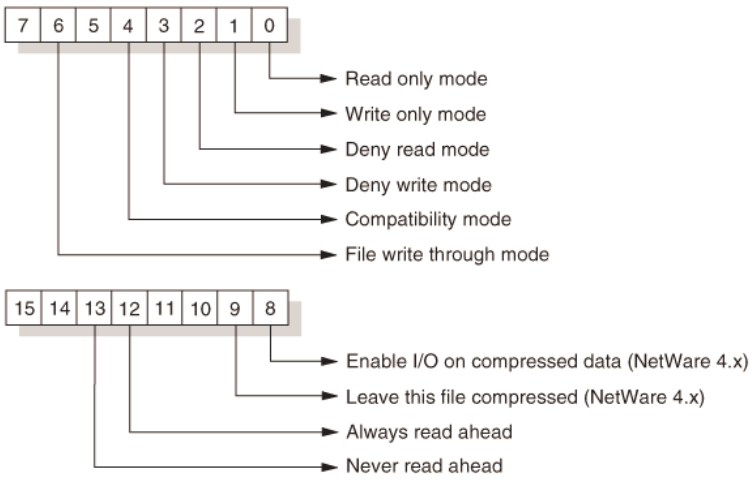
createAttributes

Contains the attributes that are to be set when the entry is created.

requestedAccessRights

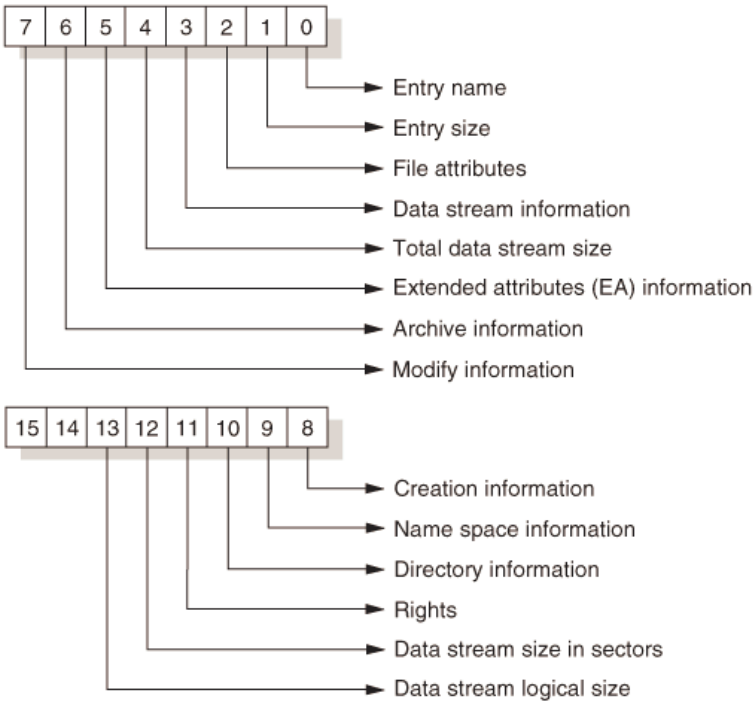
Indicates how the entry is to be opened, such as Read Only, Read Write, Compatibility mode, and so on. The bits in this mask are

defined in the following figure.



returnInfoMask

Contains a bit mask defining the information that is requested for this operation. This bit mask is shown in the following figure.



fileHandle

Contains the NetWare file handle of the entry to be created.

openCreateAction

Contains the results of the requested action (see the following table).

Value	Meaning
0x01	Opened
0x02	Created
0x04	Truncated
0x08	Compressed (for NetWare 4.0 servers only)
0xFF	Bad action

GenericPurgeDeletedCBStruct

Contains information about a generic purge deleted operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   nameSpace;
   LONG   sequence;
   LONG   volume;
   LONG   dirBase;
} GenericPurgeDeletedCBStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

sequence

Contains the NetWare-internal number that was generated while scanning for deleted files.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

GenericRenameCBStruct

Contains information about a generic rename operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   nameSpace;
   LONG   renameFlag;
   LONG   searchAttributes;
   LONG   srcVolume;
   LONG   srcPathComponentCount;
   LONG   srcDirBase;
   BYTE   *srcPathString;
   LONG   dstVolume;
   LONG   dstPathComponentCount;
   LONG   dstDirBase;
   BYTE   *dstPathString;
} GenericRenameCBStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2

5	NT

renameFlag

Contains values defining rename options, as listed in the following table.

Value	Meaning
0x01	Allow renames to myself (same name)
0x02	Rename incompatibility mode
0x04	Only change name in the entry for the specified name space

searchAttributes

Contains field contains a bit mask of the file attributes that are affected by this operation. That is, entries that have file attributes matching this bit mask are affected.

srcVolume

Contains the volume number of the entry to be renamed.

srcPathComponentCount

Contains the number of path components for the source path.

srcDirBase

Contains the source directory base.

srcPathString

Contains the path string of the source.

dstVolume

Contains the volume number of the renamed entry.

dstPathComponentCount

Contains the number of path components for the destination path.

dstDirBase

Contains the destination directory base.

dstPathString

Contains the path string of the destination.

GenericSalvageDeletedCBStruct

Contains information about a generic salvage deleted operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   nameSpace;
   LONG   sequence;
   LONG   volume;
   LONG   dirBase;
   BYTE   *newName;
} GenericSalvageDeletedCBStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

sequence

Contains the NetWare-internal number that was generated while scanning for deleted files.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

newName

Contains the new name of the file or directory.

ModifyDirEntryCallBackStruct

Contains information about a modify directory operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   pathComponentCount;
   LONG   nameSpace;
   LONG   attributeMatchBits;
   LONG   targetNameSpace;
   struct ModifyStructure *modifyVector;
   LONG   modifyBits;
   LONG   allowWildCardsFlag;
} ModifyDirEntryCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

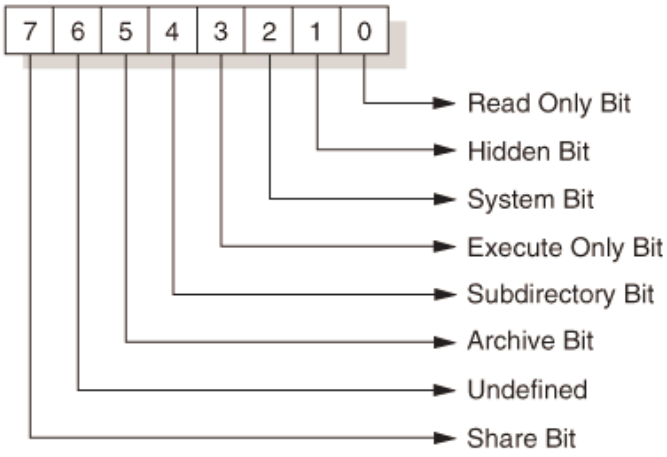
nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

attributeMatchBits

Contains a bit mask of the file attributes that are affected by this operation. That is, entries that have file attributes matching this bit mask are affected. The first byte of the file attributes mask is shown in the following figure. For more about the file attributes mask, see File Attributes.



targetNameSpace

Contains the name space of the entry that is to be changed (see the values for *nameSpace*, above).

modifyVector

Contains the modify vector used in the operation. See the discussion of ModifyStructure.

modifyBits

Contains the modify bits used in the operation. The modify bits are defined in NWDIR.H and have the following values:

```
MModifyNameBit                  1
```

```
MFileAttributesBit              2
MCreateDateBit                  4
MCreateTimeBit                  8
MOwnerIDBit                     0x10
MLastArchivedDateBit            0x20
MLastArchivedTimeBit            0x40
MLastArchivedIDBit              0x80
MLastUpdatedDateBit             0x100
MLastUpdatedTimeBit             0x200
MLastUpdatedIDBit               0x400
MLastAccessedDateBit            0x800
MInheritanceRestrictionMaskBit  0x1000
MMaximumSpaceBit                0x2000
MLastUpdatedInSecondsBit        0x4000
```

allowWildcardsFlag

Indicates whether wildcards are allowed in the pathname:

Nonzero = Wildcards allowed
0 = No wildcards allowed.

OpenFileCallBackStruct

Contains information about an open file operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   pathComponentCount;
   LONG   nameSpace;
   LONG   attributeMatchBits;
   LONG   requestedAccessRights;
   LONG   dataStreamNumber;
} OpenFileCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

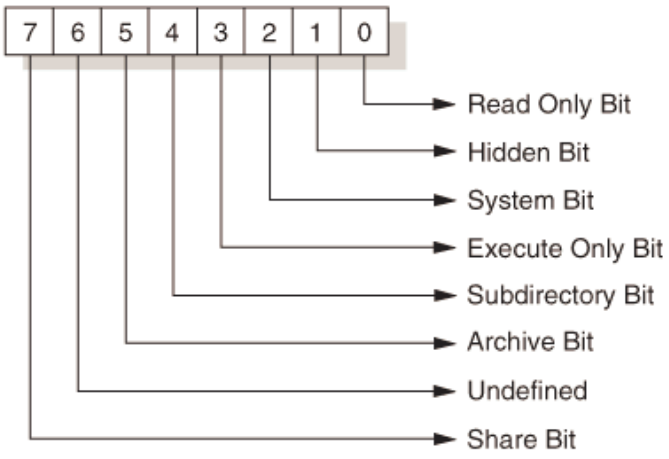
nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

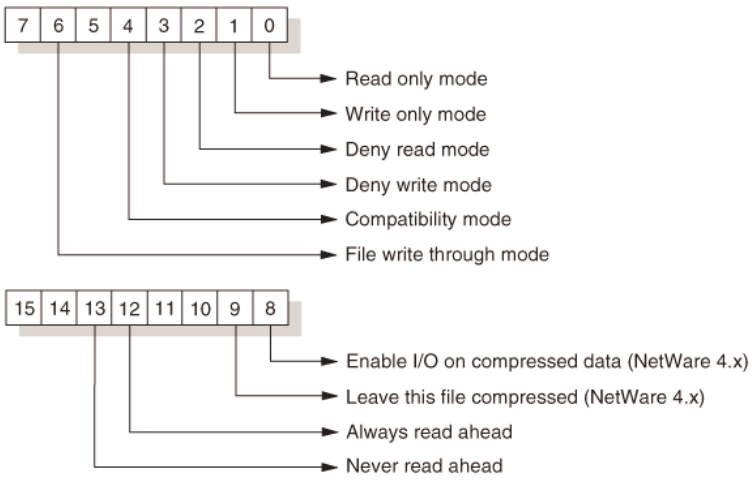
attributeMatchBits

Contains a bit mask of the file attributes that are affected by this operation. That is, entries that have file attributes matching this bit mask are affected. The first byte of the file attributes mask is shown in the following figure. For more about the file attributes mask, see File Attributes.



requestedAccessRights

Indicates how the entry is to be opened, such as Read Only, Read Write, Compatibility mode, and so on. The bits in this mask are defined in the following figure.



dataStreamNumber

Contains a number identifying the data stream type of the file or directory. The possible data streams are listed in the following table.

Number	Type
0	Primary Data Stream (corresponds to DOS)
1	Macintosh Resource Fork
2	FTAM Extra Data Fork

PurgeDeletedCallBackStruct

Contains information about a purge deleted operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   volume;
   LONG   dirBase;
   LONG   toBePurgedDirBase;
   LONG   nameSpace;
} PurgeDeletedCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the directory from which the entry is to be purged.

toBePurgedDirBase

Contains the directory base (number) that was generated while scanning for deleted files.

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2

File Service Group

RenameMoveEntryCallBackStruct

Contains information about a rename or move operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   pathComponentCount;
   LONG   nameSpace;
   LONG   attributeMatchBits;
   LONG   subDirsOnlyFlag;
   LONG   newDirBase;
   BYTE   *newPathString;
   LONG   originalNewCount;
   LONG   compatibilityFlag;
   LONG   allowRenamesToMyselfFlag;
} RenameMoveEntryCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

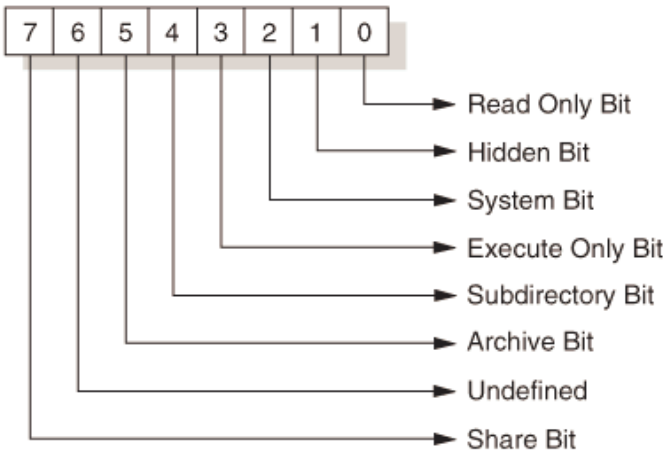
nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

attributeMatchBits

Contains a bit mask of the file attributes that are affected by this operation. That is, entries that have file attributes matching this bit mask are affected. The first byte of the file attributes mask is shown in the following figure. For more about the file attributes mask, see File Attributes.



subDirsOnlyFlag

Contains a value that indicates whether this operation is being done on a subdirectory. If the value is TRUE, the operation is being done on a subdirectory.

newDirBase

Contains the new directory base for the entry.

newPathString

originalNewCount

Contains the path count for the new path string.

compatibilityFlag

Indicates whether DOS 3.x locking compatability is to be used. If the value is TRUE, locking compatibility is to be used.

allowRenamesToMyselfFlag

Indicates whether this entry could be renamed to itself. If the value is TRUE, the entry can be renamed to itself.

RenameNSEntryCallBackStruct

Contains information about a rename name space entry operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   task;
   LONG   volume;
   LONG   dirBase;
   BYTE   *pathString;
   LONG   pathComponentCount;
   LONG   nameSpace;
   LONG   matchBits;
   BYTE   *newName;
} RenameNSEntryCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

task

Contains the task number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (directory number) of the file or directory.

pathString

Contains the NetWare-internal path string of the file or directory.

pathComponentCount

Contains the number of components in the path.

nameSpace

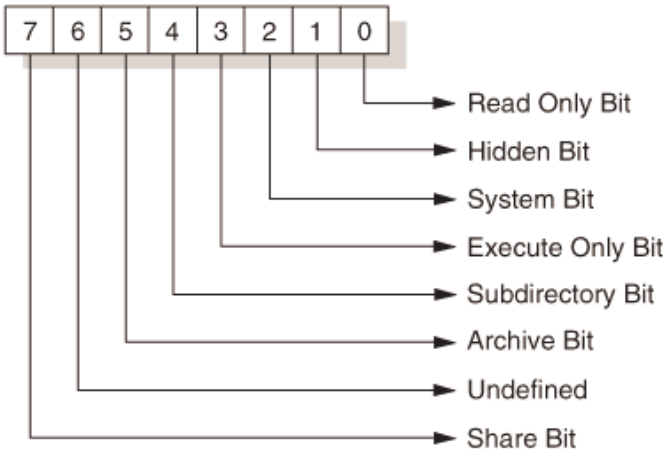
Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space

0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

matchBits

Contains a bit mask of the file attributes that are affected by this operation. That is, entries that have file attributes matching this bit mask are affected. The first byte of the file attributes mask is shown in the following figure. For more about the file attributes mask, see File Attributes.



newName

Contains the new name of the name space entry.

SalvageDeletedCallBackStruct

Contains information about a salvage deleted operation
Service: File System Monitoring

Structure

```
typedef struct {
   LONG   connection;
   LONG   volume;
   LONG   dirBase;
   LONG   toBeSalvagedDirBase;
   LONG   nameSpace;
   BYTE   *newName;
} SalvageDeletedCallBackStruct;
```

Defined In

nwfshook.h

Fields

connection

Contains the connection number of the entity requesting the operation.

volume

Contains the number of the volume that the directory entry is on.

dirBase

Contains the directory base (number) in which the entry is to be recovered to.

toBeSalvagedDirBase

Contains the directory base (number) that was generated while scanning for deleted files. This number is not the directory base that the file would be salvaged to (see *dirBase,* above).

nameSpace

Contains the name space of the file or directory. Currently defined name spaces are listed in the following table.

Value	Name Space
0	DOS
1	MACINTOSH
2	NFS
3	FTAM

4	OS2
5	NT

newName

Contains the name that the entry is to have after it is salvaged.

Media Manager

Media Manager: Guides

Media Manager: Task Guide

Parent Topic:

Media Manager: Guides

Media Manager: Concept Guide

Action Code: Insert/Remove (physical media or magazine) (0x0006)

Action Code: Label/Unlabel (physical media) (0x0009)

Action Code: Load/Unload Magazine (0x000D)

Action Code: Lock/Unlock (physical media) (0x0007)

Action Code: Mount/Dismount (physical device) (0x0004)

Action Code: Move (physical media or magazine) (0x0008)

Action Code: Scan For New Devices (adapter) (0x000A)

Action Code: Select/Unselect (physical media) (0x0005)

Action Code: Tape Control (0x0001)

Storage Access I/O Action Codes

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Locate Data Blocks (0x002B)

Action Code: Multiple File Mark (0x0027)

Action Code: Multiple Set Mark (0x0029)

Action Code: Position Media (0x002D)

Action Code: Random Read (0x0020)

Action Code: Random Write (0x0021)

Action Code: Random Write Once (0x0022)

Action Code: Reset Queue (0x0025)

Action Code: Select Partition (0x002C)

Action Code: Sequential Read (0x0023)

Action Code: Sequential Write (0x0024)

Action Code: Single File Mark (0x0026)

Action Code: Single Set Mark (0x0028)

Action Code: Space Data Blocks (0x002A)

Reserved (0x002E - 0x003F)

Code Definitions

Function/Completion Code Definitions

Alert Reasons

- Alert Types
- Application Alert Codes
- Attribute IDs
- Attribute Types
- Cartridge Types
- Console (Human Jukebox) Definitions
- Final Completion Codes
- Format Types
- Identification Types
- Initial Completion Codes
- Error Codes
- Media Types
- Message Actions
- Notify Event Bits
- Object Status Bits
- Object Types
- Reservation Modes
- Resource Tag Allocation Signatures

Functions

- Access Functions
- Human Jukebox Functions
- Identification Functions
- Information Functions
- Notification Functions
- Object Support Functions
- Partition Management Functions
- Reservation Functions
- Vendor Pass-Through Functions

Media Manager: Tasks

Media Manager: Concepts

Media Manager: Functions

Media Manager: Structures

Parent Topic:

Media Manager: Guides

Function/Completion Code Definitions

This appendix contains the public Media Manager definition statements listed alphabetically by type.

Alert Reasons

Alert Types

Application Alert Codes

Attribute IDs

Attribute Types

Cartridge Types

Console (Human Jukebox) Definitions

Final Completion Codes

Format Types

Identification Types

Initial Completion Codes

Error Codes

Media Types

Message Actions

Notify Event Bits

Object Status Bits

Object Types

Reservation Modes

Resource Tag Allocation Signatures

Parent Topic:

Media Manager: Guides

Media Manager Introduction

The Media Manager is a library of interface functions that work with the NetWare® 3.12 and 4.x OS that allow it to support many different kinds of storage devices. As a replacement to the Disk Application Interface (DAI) Specification, the Media Manager provides a more robust and comprehensive interface to storage applications and/or loadable file systems. The Media Manager interface also provides API facilities to interface with server administrators who may need to manipulate and control removable storage media.

This section discusses these aspects of Media Manager:

Media Manager Architecture

Media Manager Database Concepts

Media Manager Queue Management

Parent Topic:

Media Manager: Guides

Storage Access Control Action Codes (0x0000 - 0x001F)

Storage access action codes are divided into two mutually exclusive categories: Control action codes (this section) and I/O action codes (see " Storage Access I/O Action Codes (0x0020 - 0xFFFF)"). Control action codes typically modify the state of media objects. Consequently, these action codes must be performed serially. Also, the type of control action code available depends upon the type of the target object.

Information returned from the Control action codes comes back to the application in one of two ways, depending on whether the action code request was issued from a blocking or nonblocking function. The following table summarizes the return parameter and completion code return values for blocking and nonblocking functions:

Table auto. Comparison of Return Values for Blocking and Nonblocking Functions

Action Code	Return Parameter	Completion Code
Requesting the action from a nonblocking function (for example, **MM_Object_IO**)	Value returned in the *returnParameter*	Value returned in the *completionCode* field of the App.'s

	field of the App.'s CallBackFunction	CallBackFunction
Requesting the action from a blocking function (for example, **MM_Object_Blocking_IO**)	Value returned in the LONG * *returnParameter* field of the blocking function, etc.	Value returned as the LONG return value of the blocking function.

Action Code: Activate/Deactivate (physical media) (0x0003)

Action Code: Format (0x0000)

Action Code: Insert/Remove (physical media or magazine) (0x0006)

Action Code: Label/Unlabel (physical media) (0x0009)

Action Code: Load/Unload Magazine (0x000D)

Action Code: Lock/Unlock (physical media) (0x0007)

Action Code: Mount/Dismount (physical device) (0x0004)

Action Code: Move (physical media or magazine) (0x0008)

Action Code: Scan For New Devices (adapter) (0x000A)

Action Code: Select/Unselect (physical media) (0x0005)

Action Code: Tape Control (0x0001)

Parent Topic:

Media Manager: Guides

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

I/O action codes (described in this chapter) facilitate the movement of data to and from the media. Applications may issue multiple I/O requests concurrently. The I/O action codes that are available at any one time depend on the capabilities of the target media.

Information returned from the I/O action codes comes back to the application in one of two ways, depending on whether the action request was issued from a blocking or nonblocking function. See the table in the " Storage Access Control Action Codes (0x0000 - 0x001F)" section for a summary of the Return Parameter and Completion Code return values for blocking and nonblocking functions.

Action Code: Locate Data Blocks (0x002B)

Action Code: Multiple File Mark (0x0027)

Action Code: Multiple Set Mark (0x0029)

Action Code: Position Media (0x002D)

Action Code: Random Read (0x0020)

Action Code: Random Write (0x0021)

Action Code: Random Write Once (0x0022)

Action Code: Reset Queue (0x0025)

Action Code: Select Partition (0x002C)

Action Code: Sequential Read (0x0023)

Action Code: Sequential Write (0x0024)

Action Code: Single File Mark (0x0026)

Action Code: Single Set Mark (0x0028)

Action Code: Space Data Blocks (0x002A)

Reserved (0x002E - 0x003F)

Parent Topic:

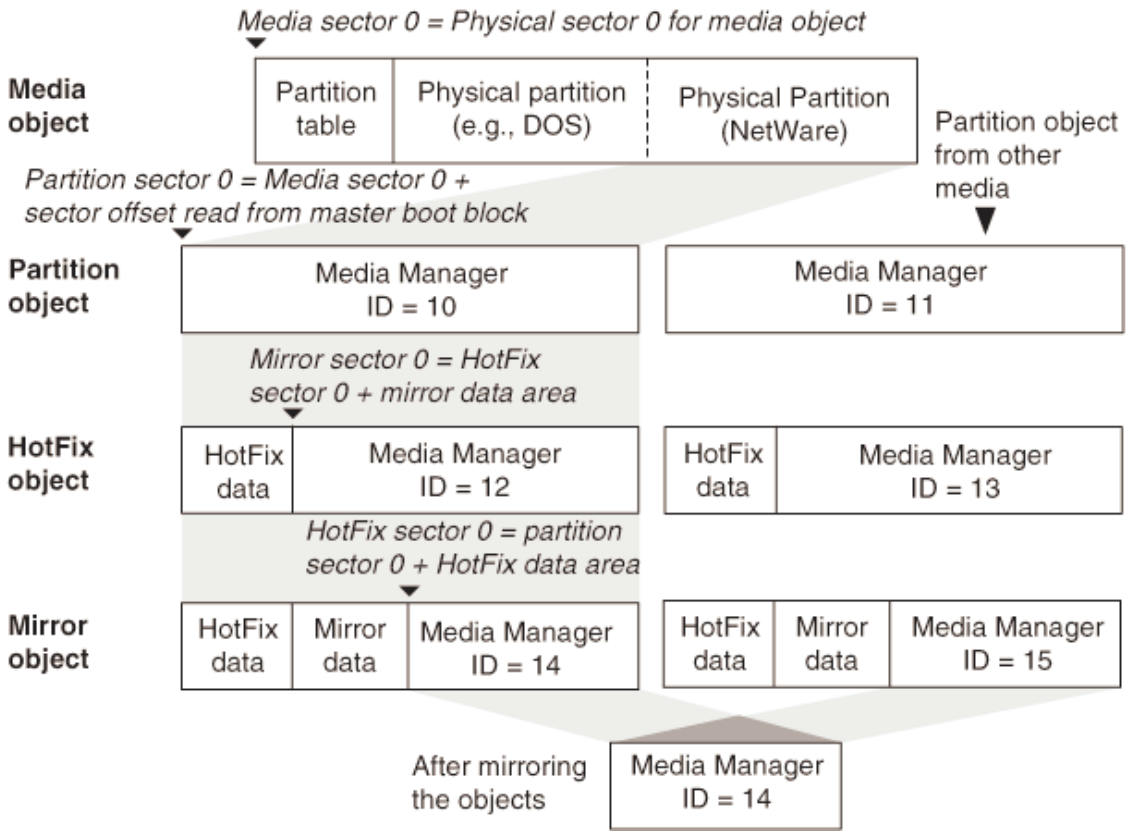
Media Manager: Guides

Media Manager: Tasks

Using Media Manager Partition Functions

Before an application developer can successfully use the Partition functions, a basic understanding of the Media Manager object database hierarchy is required. At the top of the Media Manager's object hierarchy is the Adapter Object. For every nonremovable device (i.e., a device that can contain only one piece of media, e.g., a hard disk) the Media Manager automatically creates a Media Object. The following figure shows an example of how, by starting from the two Media Objects (one of which is not shown), the Partition API is used to build a single Mirror Object. The file system will then perform I/O to the Mirror Object. Following is a sample sequence of events that properly use the Partition functions.

Figure 7. Media Manager Object Relationships to Physical Sectors



1. **Create a Partition Object**

   Each Media Object is divided into partitions using the standard IBM PC hard disk format. This scheme allows up to four partitions per media. NetWare®, however, currently allows only one NetWare partition per media. Use **MM_InitializePartitionTable** and **MM_CreatePartition** function calls to create two Partition Objects (IDs 10 and 11 in the above figure). **MM_InitializePartitionTable** is only needed if the Partition Table contains an other than IBM PC partition table and partition table signature, or the Partition Table does not currently exist (e.g. new media). The function **MM_CreatePartition** will verify the validity of the partition table before creating a Partition Object. The function **MM_DeletePartition** will remove the NetWare partition from the media and delete the Partition Object from the database.

   If other than NetWare (e.g., DOS) partitions exist they will also be children of the media object, as is shown in the above figure. Application developers should verify that they create the following objects on the NetWare partition only.This can be verified by using **MM_Return_Object_Specific_Info** for the partition object and checking the *partitionType* parameter in the PartitonInfoDef structure returned by this function. Currently, only the lower BYTE of this LONG parameter is important.

2. **Create a HotFix Object**

   HotFix Objects are created as children to previously created Partition Object using **MM_CreateHotFix**. These become the next objects (IDs 12 and 13 in the above figure) in the database. This step allocates space to be used for the HotFix area. This is currently a required step.

3. **Create a Mirror Object**

   Mirror Objects (IDs 14 and 15 in the above figure) are created as children to the previously created HotFix Objects using **MM_CreateMirror**. This step allocates space to be used for the mirror process. This step is required even if mirroring will not be performed.

4. **Group Mirror Objects into a Mirror Group (Optional)**

   If no mirroring will be performed, there is no need to continue. However, to mirror the objects one Mirror Object (ID 15 in the above figure) must be "mirrored" to another Mirror Object (ID 14 in the above figure). This is done with **MM_AddMirrorObjectToMirrorGroup**.

   > **IMPORTANT:** It is highly recommended that this function be used only on empty Mirror Objects. Performing this operation on Mirror Objects containing information can (and most likely will) cause data corruption.

   The Mirror Object identified by *mirrorObjectID* (the first parameter) is

added to the "mirror group" identified by *mirrorGroupObjectID* (the second parameter). For newly created Mirror Groups a call to **MM_ForceMirrorGroupInSync** is recommended at this time to prevent mirroring of random data that may exist in the *mirrorGroupObjectID* area. Also, for newly created Mirror Groups the mirroring direction (what gets mirrored to what) is somewhat arbitrary. However, if the application is mirroring Mirror Objects where one object contains data (see Important above), the direction is **critical**. Information from the Mirror Object *mirrorObjectID* will be overwritten with information from Mirror Object *mirrorGroupObjectID*. This process deletes the Mirror Object mirrorObjectID from Media Manager's database. Following the example in the above figure, the Mirror Object with ID = 15 will be deleted from Media Manager's database by calling **MM_AddMirrorObjectToMirrorGroup**. Any further operations to that object are performed by operations to the Mirror Object ID=14. This is why the parameter passed to **MM_DelPartitionFromMirrorGroup** identifies a Partition Object as that is the only way to identify the object to be removed from the mirror group.

Parent Topic:

Media Manager: Guides

Media Manager: Concepts

Access Functions

After the application learns all the details about the storage devices, and sets up the proper safeguards (by using notification and reservation functions), the application may then access the storage media. Using the I/O and Control functions available with Media Manager, the application can read, write, activate, deactivate, label, unlabel, etc. After completion of any function, the Media Manager will call the application's "call back" function to post completion of the event.

Also, a special "Abort Function" exists that applications can call if they have initiated a function and then need to abort it before it finishes. This facility allows applications to complete tasks without waiting for unnecessary request call backs. The Access Functions and the associated Application Function are:

MM_Abort_Function

MM_Check_For_Pending_Aborts

MM_Object_Blocking_IO

MM_Object_IO CallBackFunction (Application Function)

MM_Special_Object_Blocking_IO

Parent Topic:

Media Manager: Guides

Action Code: Activate/Deactivate (physical media) (0x0003)

Description:

The purpose of this action code is to bring the media on-line or off-line. If the media is removable, action code 0x0003 will mount or dismount the media in the device. This call is valid for both device and media objects.

Parameters:

handle

The handle of the media or device object to be mounted or

dismounted.

actionCode

0x0003 (Activate/Deactivate request)

parameter1

This parameter indicates the operation to perform:

0 activate
1 deactivate

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Format (0x0000)

Description:

This action code formats writeable media. It implies to the Media Manager that the media is new media. The media ID is implied by the object handle used.

Parameters:

handle

The handle of the object related to the logical media being formatted.

actionCode

0x0000---format request

parameter1

This parameter indicates the operation to perform:

0 quick format (re-format)
1 security erase (complete format)

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Insert/Remove (physical media or magazine) (0x0006)

Description:

The purpose of this action code is to move media or magazine objects in and out of removable devices and changers. This action code generates console alerts.

Parameters:

handle

The handle of the device.

actionCode

0x0006---insert/remove media or magazine request

parameter1

This parameter indicates the operation to perform:

0 insert
1 remove

parameter2

Contains the object ID of the media or magazine object to be inserted. A "-1" specifies new media should be inserted.

parameter3

Specifies the correct mail (exchange) slot in a media changer when there is more than one slot in the changer. This parameter should be 0 when there is only one mail slot, or if the device is a stand-alone device.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Label/Unlabel (physical media) (0x0009)

Description:

The label action code causes the Media Manager to either stamp or remove identification information on activated media. The database is updated to reflect the correct identity of the media. The media ID is derived from the object handle passed.

Parameters:

handle

The handle of an object related to the media being labeled.

actionCode

0x0009---stamp request

parameter1

This parameter indicates the operation to perform:

0 label media

1 unlabel media

parameter2

This parameter is the Novell® assigned number of the identification action code used to label the information on the media.

parameter3

Reserved. Must be set to 0.

bufferLen

The number of bytes in the label name.

LabelName

Pointer to a null-terminated length preceded string. The total bytes in

the name must not exceed 64, including the length byte and the null terminator byte.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Load/Unload Magazine (0x000D)

Description:

The purpose of this action code is to load or unload a media magazine in a device.

Parameters:

handle

The handle of device with a magazine loaded.

actionCode

0x000D---load/unload media or magazine request

parameter1

This parameter indicates the operation to perform:

0 load magazine
1 unload magazine

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Locate Data Blocks (0x002B)

Description:

This action code locates specific data blocks and returns location information.

Parameters:

handle

The handle of an object identifying the media.

actionCode

0x002B---locate data blocks

parameter1

This parameter indicates the operation to perform:

0 - return location information
1 - go to specified location

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

This parameter indicates the size of the buffer where location information is to be copied or accessed. The buffer must be of sufficient length to accommodate the location information.

buffer

If return location information is specified, this parameter points to a buffer where the location information is copied; otherwise, it contains the location information needed to locate the data block.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Lock/Unlock (physical media) (0x0007)

Description:

The purpose of this action code is to lock media that has been loaded in a device. Once locked, the media cannot be manually ejected from the device by the operator via the use of the front panel eject switch (if the device is so equipped).

Parameters:

handle

The handle of a media or magazine object.

actionCode

0x0007---lock/unlock media or magazine request

parameter1

This parameter indicates the operation to perform:

0 lock media selection
1 unlock media selection

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Mount/Dismount (physical device) (0x0004)

Description:

The purpose of this action code is to bring the media inside a physical device on-line or off-line. The mount/dismount action code assumes that the media is present. This operation will identify or verify the media in the device.

> **NOTE:** This action code will not return until all notifications have been started; however, it will not wait for the notifications to complete before returning.

before returning.

Parameters:

handle

The handle of the media or device object to be mounted or dismounted.

actionCode

0x0004---mount/dismount request

parameter1

This parameter indicates the operation to perform:

0 mount
1 dismount

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Move (physical media or magazine) (0x0008)

Description:

The purpose of this action code is to move physical media or magazine objects inside a media changer.

NOTE: The source location is implied by the media or magazine object.

Parameters:

handle

The handle of a media or magazine object.

actionCode

0x0008---Move media or magazine request

parameter1

This parameter is the object ID of the media inside a media changer.

parameter2

The slot number of an empty slot or device in a media changer.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Multiple File Mark (0x0027)

Description:

This action code performs the writing and locating of multiple, contiguous file marks.

Parameters:

handle

The handle of an object identifying the media

actionCode

0x0027---multiple file marks

parameter1

This parameter indicates the operation to perform:

0 - write multiple file marks.

1 - space forward for multiple file marks.

2 - space backward for multiple file marks.

parameter2

This parameter indicates the number of consecutive file marks to be written to or spaced over.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Multiple Set Mark (0x0029)

Description:

This action code performs the writing and locating of multiple, contiguous set marks.

Parameters:

handle

The handle of an object identifying the media

actionCode

0x0029---multiple set marks

parameter1

This parameter indicates the operation to perform:

0 - write multiple set marks
1 - space forward for multiple set marks.
2 - space backward for multiple set marks.

parameter2

This parameter indicates the number of consecutive set marks to be written to or spaced over.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Position Media (0x002D)

Description:

This action code provides support to position sequential media devices.

Parameters:

handle

The handle of an object identifying the media

actionCode

0x002D---sequential media

parameter1

This parameter indicates the operation to perform:

1 rewind media.

2 go to end of recorded media (logical end of media).

parameter2

Reserved. Must be set to 0

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Random Read (0x0020)

Description:

This action code performs random read I/O from the device.

Parameters:

handle

The handle of an object related to the media being read.

actionCode

0x0020---read request

parameter1

This parameter specifies the number of units to be read.

parameter2

This parameter specifies the logical unit number of the beginning unit.

parameter3

Reserved. Must be set to 0.

bufferLen

This parameter is the size of the buffer (i.e. the number of units * the unit size).

buffer

This parameter points to a buffer in memory were the data is to be read.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Random Write (0x0021)

Description:

This action code writes to any part of the media in a device.

Parameters:

handle

The handle of an object related to the media being written.

actionCode

0x0021---write request

parameter1

This parameter specifies the number of units to be written.

parameter2

This parameter specifies the logical unit number of the beginning unit.

parameter3

Reserved. Must be set to 0.

bufferLen

This parameter is the size of the buffer (i.e. the number of units * the unit size).

buffer

This parameter points to a buffer in memory were the data is written from.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Random Write Once (0x0022)

Description:

This action code performs one write to any part of the media in a device.

Parameters:

handle

The handle of an object related to the media being written.

actionCode

0x0022---write once request

parameter1

This parameter specifies the number of units to be written.

parameter2

This parameter specifies the logical unit number of the beginning unit.

parameter3

Reserved. Must be set to 0.

bufferLen

This parameter is the size of the buffer (i.e. the number of units * the unit size)

buffer

This parameter points to a buffer in memory were the data is written

from.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Reset Queue (0x0025)

Description:

This action code resets the end of tape status in the device so that data can be written. Once the driver detects an early warning signal, it rejects any write commands until it receives a Reset Queue command from the application. After receiving the Reset Queue command, the driver will write only once before it needs to receive another Reset Queue. This call insures that the driver will at least write one more request before running out of tape.

This action code also restarts the queue for the device under the NWPA architecture.

Parameters:

handle

The handle of an object related to the media to be reset.

actionCode

0x0025---reset queue

parameter1

Reserved. Must be set to 0.

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Scan For New Devices (adapter) (0x000A)

Description:

This action code causes the device driver to look for new devices that might have come on-line.

Parameters:

handle

The handle of an adapter object.

actionCode

0x000A---scan for new devices request

parameter1

Reserved. Must be set to 0.

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Select Partition (0x002C)

Description:

This action code selects a tape partition and positions the media at the beginning or end of the tape partition depending on parameter 1.

Parameters:

handle

The handle of an object identifying the media

actionCode

0x002C---select partition

parameter1

This parameter indicates the operation to perform:

1 rewind/select tape partition.
2 go to end of file/select tape partition.

parameter2

This parameter indicates the tape partition to be selected.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Select/Unselect (physical media) (0x0005)

Description:

The purpose of this action code is to select or unselect an individual media from a media magazine to be inserted into the device. It assumes that the magazine is loaded in a device.

NOTE: A selection of a media implies that the previously selected media is no longer selected.

Parameters:

handle

The handle of the device, magazine or media object to be dismounted.

actionCode

0x0005---select/unselect

parameter1

This parameter indicates the operation to perform:

0 select
1 unselect

parameter2

The slot number of the slot containing the media to be moved ("selected") into a device, **or** the slot number of an empty slot in the magazine where the media will go when it is moved ("unselected") from the device.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Code: Sequential Read (0x0023)

Description:

This action code reads sequential data from tape.

Parameters:

handle

The handle of the object representing the media to be read.

actionCode

0x0023---read request

parameter1

This parameter specifies the number of units to be read, where units refers to what was formerly called sectors on fixed media.

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

This parameter is the size of the buffer (i.e. the number of units * the unit size) in bytes. The "number of units" (parameter1) cannot exceed the maximum number of units as prescribed in the *blockSize* returned in a GenericInfoDef structure. `unitSize` is also returned in a GenericInfoDef structure.

buffer

Points to a buffer in memory where the data is to be read.

Return Values:

returnParameter

The number of units read. (See the table in Storage Access Control Action Codes (0x0000 - 0x001F) for more specific information.)

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Sequential Write (0x0024)

This action code performs sequential write I/O to the device.

Parameters:

handle

The handle of an object related to the media being written.

actionCode

0x0024---write request

parameter1

This parameter specifies the number of units to be written.

parameter2

Reserved. Must be set to 0.

parameter3

Reserved. Must be set to 0.

bufferLen

This parameter is the size of the buffer (i.e. the number of units * the unit size).

buffer

This parameter points to a buffer in memory where the data is written from.

Return Values:

returnParameter

The number of units written. (See the table in Storage Access Control Action Codes (0x0000 - 0x001F) for more specific information.)

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Single File Mark (0x0026)

Description:

This action code performs the writing and locating of single file marks.

Parameters:

handle

The handle of an object identifying the media.

actionCode

0x0026---single file mark

parameter1

This parameter indicates the operation to perform:

0 - write single file mark
1 - space forward single file marks
2 - space backwards single file marks

parameter2

If a space action code was requested, this parameter indicates how many file marks to space over.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

If a "space" setting (1 or 2) was selected in *parameter1*, the number of file marks spaced over is returned in this field (see the table in Storage Access Control Action Codes (0x0000 - 0x001F) for more information.)

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Single Set Mark (0x0028)

Description:

This action code performs the writing and locating of single set marks.

Parameters:

handle

The handle of an object identifying the media.

actionCode

0x0028---single set mark

parameter1

This parameter indicates the operation to perform:

0 - write single set mark

1 - space forward single set marks

2 - space backward for single set marks

parameter2

If a space action code was requested, this parameter indicates how many set marks to space over.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

If a "space" setting (1 or 2) was selected by *parameter1*, the number of set marks spaced over is returned in this field (see the table inStorage Access Control Action Codes (0x0000 - 0x001F) for more information.)

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Space Data Blocks (0x002A)

Description:

This action code spaces over data blocks.

Parameters:

handle

The handle of an object identifying the media.

actionCode

0x002A---space data blocks

parameter1

This parameter indicates the direction of the space:

1 - space forward for multiple data blocks.
2 - space backward for multiple data blocks.

parameter2

This parameter indicates the number of data blocks to be spaced over.

parameter3

Reserved. Must be set to 0.

bufferLen

Must be set to 0.

buffer

Must be set to 0.

Return Values:

returnParameter

The parameter should return the actual number of data blocks spaced over (see the table in Storage Access Control Action Codes (0x0000 - 0x001F) for more information.)

Parent Topic:

Storage Access I/O Action Codes (0x0020 - 0xFFFF)

Action Code: Tape Control (0x0001)

Description:

Provides additional action codes to manage tapes. Use various combinations of parameters to create tape partitions, retention, and perform two kinds of erase partition operations. (Note: the partitions referred to here are **not** NetWare® partitions.)

Both erase action codes take place from the current position to the end of the tape partition. They **do not** imply a rewind of tape media.

At the end of the retention operation, the current tape partition is undefined. Select another partition (See Action Code: Select Partition (0x002C)) before continuing.

Parameters:

handle

The handle of the device received from **MM_Reserve_Object**.

actionCode

0x0001---tape operation request

parameter1

This parameter indicates the operation to perform:

0 quick erase (re-format)
1 security erase (zero out all blocks)
2 create tape partition
3 retention

parameter2

Indicates the number of tape partitions to create.(only used if Parameter 1 = create tape partition).

parameter3

Reserved. Must be set to 0.

bufferLen

Byte count of buffer array; equals (Parameter 2) * 4.

buffer

Pointer to array of LONGs, where each LONG indicates the size of each tape partition in megabytes. A "-1" in an array location indicates the tape partition should take the remainder of the tape. The last partition should always have a "-1".

> **NOTE:** Partition 0 is always the last partition on the tape and will be the first LONG in this array of LONGS.

Return Values:

returnParameter

Not used.

Parent Topic:

Storage Access Control Action Codes (0x0000 - 0x001F)

Action Codes

These are control and I/O action codes that applications employ to communicate with the storage device. The control action codes (seeStorage Access Control Action Codes (0x0000 - 0x001F)) generally perform tasks that do not require the movement of data to and from the device. The I/O action codes (see Storage Access I/O Action Codes (0x0020 - 0xFFFF)) perform writes, reads, resets, and positioning operations.

Control Action Codes

```
Activate/Deactivate
Format
Insert/Remove
Label/Unlabel
Load/Unload Magazine
Lock/Unlock
Mount/Dismount
Move
Scan For New Devices
Select/Unselect
Tape Control
```

I/O Action Codes

```
Locate Data Blocks
Multiple File Mark
Multiple Set Mark
Position Media
Random Read
Random Write
Random Write Once
Reset Queue
Select Partition
Sequential Read
Sequential Write
Single File Mark
Single Set Mark
Space Data Blocks
```

Parent Topic:

Media Manager: Guides

Alert Reasons

Used by the AlertFunction defined in **MM_Reserve_Object**.

```
#define ALERT_HOTFIX_ERROR                      0x00000000
#define ALERT_DRIVER_UNLOAD                     0x00000001
#define ALERT_DEVICE_FAILURE                    0x00000002
#define ALERT_PROGRAM_CONTROL                   0x00000003
#define ALERT_MEDIA_DISMOUNT                    0x00000004
#define ALERT_MEDIA_EJECT                       0x00000005
#define ALERT_SERVER_DOWN                       0x00000006
#define ALERT_SERVER_FAILURE                    0x00000007
#define ALERT_MEDIA_LOAD                        0x00000008
#define ALERT_MEDIA_MOUNT                       0x00000009
#define ALERT_DRIVER_LOAD                       0x0000000A
#define ALERT_LOST_SOFTWARE_FAULT_TOLERANCE     0x0000000B
```

```
#define ALERT_INTERNAL_OBJECT_DELETE           0x0000000C
#define ALERT_MAGAZINE_LOAD                    0x0000000D
#define ALERT_MAGAZINE_UNLOAD                  0x0000000E
#define ALERT_DEVICE_GOING_TO_BE_REMOVED       0x0000000F
#define ALERT_CHECK_DEVICE                     0x00000010
#define ALERT_CONFIGURATION_CHANGE             0x00000011
#define ALERT_APPLICATION_UNREGISTER           0x00000012
#define ALERT_DAI_EMULATION                    0x00000013
#define ALERT_LOST_HARDWARE_FAULT_TOLERANCE    0x00000014
#define ALERT_INTERNAL_OBJECT_CREATE           0x00000015
#define ALERT_INTERNAL_MANAGER_REMOVE          0x00000016
#define ALERT_DEVICE_GOING_TO_BE_DEACTIVATED   0x00000017
#define ALERT_DEVICE_END_OF_MEDIA              0x00000018
#define ALERT_MEDIA_INSERTED                   0x00000019
#define ALERT_UNKNOWN_DEVICE_ALERT             0x0000001A
#define ALERT_UNKNOWN_ADAPTER_ALERT            0x0000001B
#define ALERT_BASE_FILTER_REMOVED              0x0000001C
```

Parent Topic:

Function/Completion Code Definitions

Alert Types

Used by the AlertFunction defined in **MM_Reserve_Object**.

```
#define ALERT_MESSAGE             0x00000001
#define ALERT_ACTIVATE            0x00000002
#define ALERT_DEACTIVATE          0x00000003
#define ALERT_DELETE              0x00000004
#define ALERT_PARENT_MESSAGE      0x00000005
#define ALERT_PARENT_ACTIVATE     0x00000006
#define ALERT_PARENT_DEACTIVATE   0x00000007
#define ALERT_PARENT_DELETE       0x00000008
```

Parent Topic:

Function/Completion Code Definitions

Application Alert Codes

Used by **MM_Set_Unload_Semaphore**.

```
#define GOING_TO_BE_DEACTIVATED       0x0001
#define OBJECT_BEING_DEACTIVATED      0x0002
#define OBJECT_SIZE_CHANGED           0x0003
#define OBJECT_BEING_ACTIVATED        0x0004
#define OBJECT_BEING_DELETED          0x0005
#define OBJECT_LOST_FAULT_TOLERANCE   0x0006
```

Parent Topic:

Function/Completion Code Definitions

Attribute IDs

Used in **MM_Return_Object_Attribute**, **MM_Return_Objects_Attributes**, and **MM_Set_Object_Attribute**.

```
MM_LABEL = 42414C05
MM_CARTRIDGE_TYPE = 5241430E
MM_UNIT_SIZE = 494E5509
MM_BLOCK_SIZE = 4F4C420A
MM_CAPACITY = 50414308
MM_PREFERRED_SIZE = 4552500E
MM_NAME = 4D414E04
MM_TYPE = 50595404
MM_SLOT_NUMBER = 4F4C530B
MM_REMOVABLE = 4D455209
MM_READ_ONLY = 41455209
MM_MIRROR_COUNT = 52494D0C
MM_MIRROR_OPERATIONAL = 52494D12
MM_MIRROR_INSYNCH = 52494D0E
MM_MIRROR_GROUP_PRESENT = 52494D14
MM_MIRROR_ORPHAN = 52494D0D
MM_REMIRROR_STATUS = 4D45520F
MM_REMIRROR_PERCENT = 4D455210
MM_PARTITION_TYPE = 5241500E
MM_PARTITION_OFFSET = 52415010
MM_PARTITIONER_TYPE = 52415010
MM_HOTFIX_SIZE = 544F480B
MM_HOTFIX_AVAILABLE = 544F4810
MM_HOTFIX_BLOCKS = 544F480D
MM_HOTFIX_SYSTEM_BLOCKS = 544F4814
MM_RAW_DEVICE_NAME = 5741520F
MM_HEADS = 41454805
MM_SECTORS = 43455307
MM_CYLINDERS = 4C594309
MM_LUN = 4E554C03
MM_CONTROLLER = 4E4F430A
MM_CARD = 52414304
MM_DRIVER = 49524406
MM_DRIVER_NAME = 4952440B
MM_SLOT = 4F4C5304
MM_PORT = 524F5004
MM_MEMORY = 4D454D06
MM_INTERRUPT = 544E4909
MM_DMA = 414D4403 /* Attribute IDs  (Continued) */
MM_TAPE_POSITION_SIZE = 50415412
MM_TAPE_MEDIA_TYPE = 5041540F
MM_TAPE_WRITE_FORMAT = 50415411
```

```
MM_TAPE_WRITE_FORMAT = 50415411
MM_TAPE_READ_FORMAT = 50415410
MM_MINIMUM_BLOCK_SIZE = 4E494D12
MM_MAXIMUM_BLOCK_SIZE = 58414D12
MM_MAXIMUM_NUMBER_OF_PARTITIONS = 58414D1C
MM_MAXIMUM_PARTITION_SIZE = 58414D16
MM_DATA_COMPRESSION_INFO = 54414415
```

Parent Topic:

Function/Completion Code Definitions

Attribute Types

Used by `AttributeInfoDef` structure

```
MM_STRING  1   /* Byte  length preceded and null terminated string */
MM_BYTE    2
MM_WORD    3
MM_LONG    4
MM_OTHER   5/* The attribute size field in the AttributeInfoDef structu
```

Parent Topic:

Function/Completion Code Definitions

Cartridge Types

Used in GenericInfoDef

```
0x00   Fixed Media
0x01   5.25 inch Floppy
0x02   3.5 inch Floppy
0x03   5.25 inch Optical
0x04   3.5 inch Optical
0x05   .5 inch Tape
0x06   .25 inch Tape
0x07   8 mm Tape
0x08   4 mm Tape
0x09   Bernoulli Disk
```

Parent Topic:

Function/Completion Code Definitions

Completion Codes

With the current version of the Media Manager and Partition Management functions, any nonzero completion code is a fatal error for that function.

Also, the MM_FAILURE completion code (0x0A) returned by each of these functions is a generic code which indicates that, for some reason, the function failed to perform the desired operation. In future versions of NetWare®, the Media Manager functions will provide more detailed fault isolation information through the use of additional completion codes. This will assist the application developer in providing more fault-tolerant applications.

Parent Topic:

Media Manager: Guides

Console (Human Jukebox) Definitions

Used by Human Jukebox functions.

```
#define HJ_INSERT_MESSAGE   0
#define HJ_EJECT_MESSAGE    1
#define HJ_ACK_MESSAGE      2
#define HJ_NACK_MESSAGE     3
#define HJ_ERROR            4
```

Parent Topic:

Function/Completion Code Definitions

Design Goals

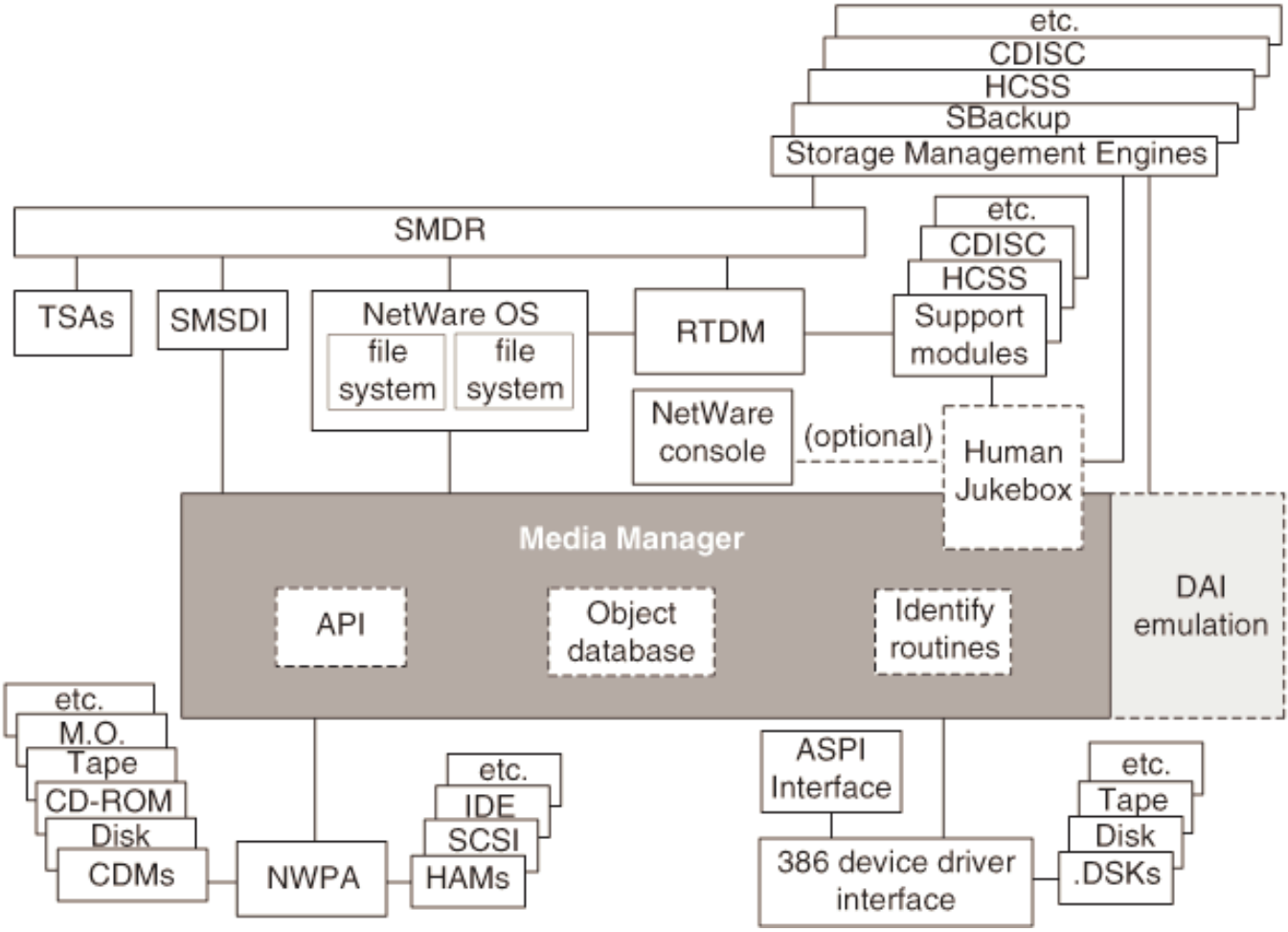
The following are goals that relate to the design of the Media Manager.

Better User Interface for Storage Management

The Media Manager provides a better user interface for server administrators and operators. The "Human Jukebox" functions give developers greater information and control over storage resources. With applications developed using the "Human Jukebox" interface, administrators can control the movement of changer media, mount and dismount media at a device level, specify media labels, and assign storage resources to specific applications.

Media Manager serves as an OS layer that insulates NLM™ applications from device driver details, as shown in the following figure.

Figure 8. Media Manager as Insulation between NLM Applications and Device Drivers



Storage Access Interface

The Media Manager provides a new storage access interface that allows applications to use all storage resources available on the system. The interface is common for all types of storage objects. It supports database queries, storage resource reservation, and storage access functions.

Media Format Independence

The Media Manager is media format independent and makes no assumptions about the layout of information on the media. It integrates application support functions to label, unlabel, and identify media.

Backward and Forward Compatibility

The Media Manager assumes all responsibilities associated with NetWare® 3.12 and 4.x compatibility issues. The following list represents areas of the OS that the Media Manager has addressed:

DAI

The Media Manager is a replacement for current DAI functionality. The Media Manager will emulate existing DAI interfaces so that current DAI-based applications can operate unmodified in the new environment. All newly developed applications should use the Media Manager interface exclusively. Any existing DAI-based applications should be updated to support the Media Manager functions defined in this document in place of the DAI interface.

International Language Support

The Media Manager is enabled using Novell® internal localization guidelines. All international language modules are fully compatible with the Media Manager.

System Fault Tolerance (SFT™) III

The Media Manager and driver interfaces will be fully compatible with SFT III™ server products. The functions defined in the DAI are also conducive to mirrored server operation.

NetWare C Library Interface (CLIB)

The Media Manager is compatible with new CLIB functions and will be compatible with future changes to the library.

Memory Protection

The Media Manager kernel and API functions are all compatible with the memory protection features that are new to the NetWare 4.x server environment.

New Storage Technology Integration

The Media Manager provides a driver interface that allows media changers to be integrated into the server. NetWare's new NetWare Peripheral

Architecture (NWPA) allows flexibility in separating the functions of the host adapter from individual device behaviors. This allows rapid integration of new storage technologies into the server. The current Device Driver specification (recently updated for NetWare 3.12 and 4.x) will continue to be supported, however.

Network Administrator Management

The Media Manager is instrumented for the NetWare Management architecture. Network management utilities and the "Human Jukebox" interface allow network administrators to obtain the state of Media Manager objects and manipulate the various storage resources.

Parent Topic:

Media Manager: Guides

Error Codes

(These are return codes for non-I/O functions.)

```
#define MM_OK                          0x00
#define MM_INVALID_OBJECT              0x01
#define MM_INVALID_APPLICATION         0x02
#define MM_INVALID_RESOURCETAG         0x03
#define MM_MEMORY_ALLOCATION_ERROR     0x04
#define MM_INVALID_MODE                0x05
#define MM_RESERVATION_CONFLICT        0x06
#define MM_PARAMETER_ERROR             0x07
#define MM_OBJECT_NOT_FOUND            0x08
#define MM_ATTRIBUTE_NOT_SETABLE       0x09
#define MM_FAILURE                     0x0A
#define MM_NOT_ABORTED                 0x0B
#define MM_IO_ERROR                      0x0C
```

Parent Topic:

Function/Completion Code Definitions

Final Completion Codes

Used by **MM_Object_IO**

NOTE: Two words are returned for each of these codes. The upper word is the return code from the adapter driver (not defined here). This list contains the lower word of the return code. The application must mask off the upper word to check for the following completion codes.

```
#define FUNCTION_OK                             0x00
#define FUNCTION_CORRECTED_MEDIA_ERROR          0x10
#define FUNCTION_MEDIA_ERROR                    0x11
#define FUNCTION_DEVICE_ERROR                   0x12
#define FUNCTION_ADAPTER_ERROR                  0x13
#define FUNCTION_NOT_SUPPORTED_BY_DEVICE        0x14
#define FUNCTION_NOT_SUPPORTED_BY_DRIVER        0x15
#define FUNCTION_PARAMETER_ERROR                0x16
#define FUNCTION_MEDIA_NOT_PRESENT              0x17
#define FUNCTION_MEDIA_CHANGED                  0x18
#define FUNCTION_PREVIOUSLY_WRITTEN             0x19
#define FUNCTION_MEDIA_NOT_FORMATED             0x1A
#define FUNCTION_BLANK_MEDIA                    0x1B
#define FUNCTION_END_OF_MEDIA                   0x1C  /* or end of parti
#define FUNCTION_FILE_MARK_DETECTED             0x1D
#define FUNCTION_SET_MARK_DETECTED              0x1E
#define FUNCTION_WRITE_PROTECTED                0x1F
#define FUNCTION_OK_EARLY_WARNING               0x20
#define FUNCTION_BEGINNING_OF_MEDIA             0x21
#define FUNCTION_MEDIA_NOT_FOUND                0x22
#define FUNCTION_MEDIA_NOT_REMOVED              0x23
#define FUNCTION_UNKNOWN_COMPLETION             0x24
#define FUNCTION_DATA_MISSING                   0x25
#define FUNCTION_HOTFIX_ERROR                   0x26
#define FUNCTION_HOTFIX_UPDATE_ERROR            0x27
#define FUNCTION_IO_ERROR                       0x28
#define FUNCTION_CHANGER_SOURCE_EMPTY           0x29
#define FUNCTION_CHANGER_DEST_FULL              0x2A
#define FUNCTION_CHANGER_JAMMED                 0x2B
#define FUNCTION_MAGAZINE_NOT_PRESENT           0x2D
#define FUNCTION_MAGAZINE_SOURCE_EMPTY          0x2E
#define FUNCTION_MAGAZINE_DEST_FULL             0x2F
#define FUNCTION_MAGAZINE_JAMMED                0x30
#define FUNCTION_ABORT_CAUSED_BY_PRIOR_ERROR    0x31
#define FUNCTION_CHANGER_ERROR                  0x32
#define FUNCTION_MAGAZINE_ERROR                 0x33
#define FUNCTION_BLOCK_SIZE_ERROR               0x34
#define FUNCTION_COMPRESSED_DATA                   0x35
```

Parent Topic:

Function/Completion Code Definitions

Format Types

(Retrieved by use of Attribute IDs)

1/4 Inch

```
QIC-24      0x00000001
QIC-120     0x00000002
```

```
QIC-120      0x00000002
QIC-150      0x00000004
QIC-320      0x00000008
QIC-525      0x00000010
QIC-1350     0x00000020
QIC-2100C    0x00000040
QIC-1000      0x00000080
```

1/2 Inch

```
X3B5/87-099    0x00000001
X3B5/86-199    0x00000002
HI-TC1         0x00000004
HI-TC2         0x00000008
X3.193-1990    0x00000010
X3B5/91-174    0x00000020
X3B5/91-227    0x00000040
X3.266-199x    0x00000080
X3B5/94-354    0x00000100
```

8 mm

```
X3.202-1991        0x00000001
ECMA TC17          0x00000002
EXABYTE-8500/05     0x00000004
```

DAT

```
DDS         0x00000001
Data DAT    0x00000002
DDS2        0x00000004
```

Parent Topic:

Function/Completion Code Definitions

Future Design Improvements

The following are features planned to be designed into future versions of the Media Manager interface, but are not part of this version of the Media Manager specification.

Static Database of Managed Objects

The static database will allow the Media Manager to keep information statically on disk about on-line media, off-line media, devices, and changers. This database will then be used to restore a server's storage configuration when a server is restarted.

On-demand Mounting

The Media Manager interface will allow a group of media to appear on-line

with the Media Manager although the group of media might be off-line. This feature will therefore automatically mount and dismount individual pieces of media on demand.

Directory Services Integration

Integrating Media Manager functionality with Directory Services requires more research. A possible application may be storing information about certain storage resources in the global directory. Where duplicate copies of static data exist, such as CD-ROM, the directory could show where the most convenient copy exits.

Parent Topic:

Media Manager: Guides

Human Jukebox Functions

The current suite of Human Jukebox functions consists of one API function and one function that the application writes to handle operator messages at the console. Both functions are optional. The human jukebox interface is used by the Media Manager to communicate with the console operator. The operator can be prompted to insert or remove media from a particular storage device and therefore become part of the storage management system as a "human jukebox."The application's optional function would provide its own screen, screen prompts and I/O to the operator. If the application does not provide this function, the Media Manager prompts the console operator on the system default server console screen. The Human Jukebox functions are:

HJ_Media_Request

HJ_Media_Request_Ack

Parent Topic:

Media Manager: Guides

Identification Functions

Applications often need to identify storage media with a customized token or label. Using labels, application developers often implement more complex functionality than would be available through the device itself. Using Media Manager's identification functions, applications can label, unlabel, and otherwise identify storage media to provide enhanced functionality. Through the reservation functions mentioned above, an application lets the Media Manager know about its functions that will

handle labeling, unlabeling, etc. of the media. The Identification functions and the associated Application functions are:

MM_Register_Identification_Routines

IdentifyFunction (Application Function)

LabelFunction (Application Function)

UnlabelFunction (Application Function)

MM_Unregister_Identification_Routines

Note that the three functions to be provided by the application are specified as part of **MM_Register_Identification_Routines**.

> **NOTE:** Developers using these functions should see Format Types for the appropriate data format type to support. If a new format type definition is needed, please contact Novell Labs™ at Novell®, Inc., 122 E. 1700 South, Provo, UT. USA 84601.

Parent Topic:

Media Manager: Guides

Identification Types

Used by `MediaInfoDef` and **MM_Register_Identification_Routines**

```
#define UNIDENTIFIABLE_MEDIA        0x00000001
#define HIGH_SIERRA_CDROM_MEDIA     0x00000002
#define ISO_CDROM_MEDIA             0x00000003
#define NETWARE_FILE_SYSTEM_MEDIA   0x00000005
#define INTERNAL_IDENTIFY_TYPE      0x00000007
#define MEDIA_TYPE_SMS              0x00000008
#define MEDIA_TYPE_SIDF             0x00000009
#define MEDIA_TYPE_BLANK            0x0000000A
#define MEDIA_TYPE_ERROR            0x0000000B
#define FORMAT_MEDIA                0x0000
#define TAPE_CONTROL                0x0001
(reserved)                          0x0002
#define ACTIVATE_FUNCTIONS          0x0003
#define MOUNT_FUNCTIONS             0x0004
#define SELECT_FUNCTIONS            0x0005
#define LOAD_FUNCTIONS                0x0006
```

Parent Topic:

Function/Completion Code Definitions

Information Functions

The information functions give applications the ability to find the details of various storage devices. Because Media Manager has a generalized architecture to support many types of devices, the detailed information needed to communicate with the device drivers is not known in advance.

After searching for the types of devices already registered with the Media Manager, an application (such as Novell's SBACKUP), would continue querying the database for generic information that all objects would have. This information includes number of children, number of siblings, and number of parents. More detailed information would include object IDs of all the related objects in the database. All this information is kept by the application while the server and the application are running. The Information functions are:

MM_Find_Object_Type

MM_Return_Object_Attribute

MM_Return_Object_Generic_Info

MM_Return_Object_Mapping_Info

MM_Return_Object_Specific_Info

MM_Return_Object_Table_Size

MM_Return_Objects_Attributes

MM_Set_Object_Attribute

Parent Topic:

Media Manager: Guides

Initial Completion Codes

These return codes are for the I/O function **MM_Object_IO**.

```
#define MESSAGE_PROCESSED                0x00
#define MESSAGE_DATA_MISSING             0x01
#define MESSAGE_POSTPONE                 0x02
#define MESSAGE_ABORTED                  0x03
#define MESSAGE_INVALID_PARAMETERS       0x04
#define MESSAGE_OBJECT_NOT_ACTIVE        0x05
#define MESSAGE_INVALID_OBJECT           0x06
#define MESSAGE_FUNCTION_NOT_SUPPORTED   0x07
#define MESSAGE_INVALID_MODE             0x08
#define MESSAGE_ABORTED_CLEAN              0x0A
```

Parent Topic:

Function/Completion Code Definitions

Media Manager Architecture

The NetWare® server architecture before 3.12 used a device manager only designed for the needs of the internal NetWare file system. Although it provided simple device level services, it did not support new storage technologies such as media changers and CD-ROM magazines or "jukebox" devices. NetWare 3.12 and 4.x OS use the Media Manager to provide a richer API set for applications to access storage media.

The main features of the Media Manager architecture include:

An interface to control and manage changers and removable devices.

Providing media identification services to storage applications.

Standard media insertion/ejection services via the "Human" jukebox.

Resource assignment and administrative services.

Object oriented design for easy maintenance and expansion.

The architecture of the Media Manager provides a foundation for more sophisticated file systems and storage applications. The Media Manager is a critical component of the future loadable file system architecture.

Parent Topic:

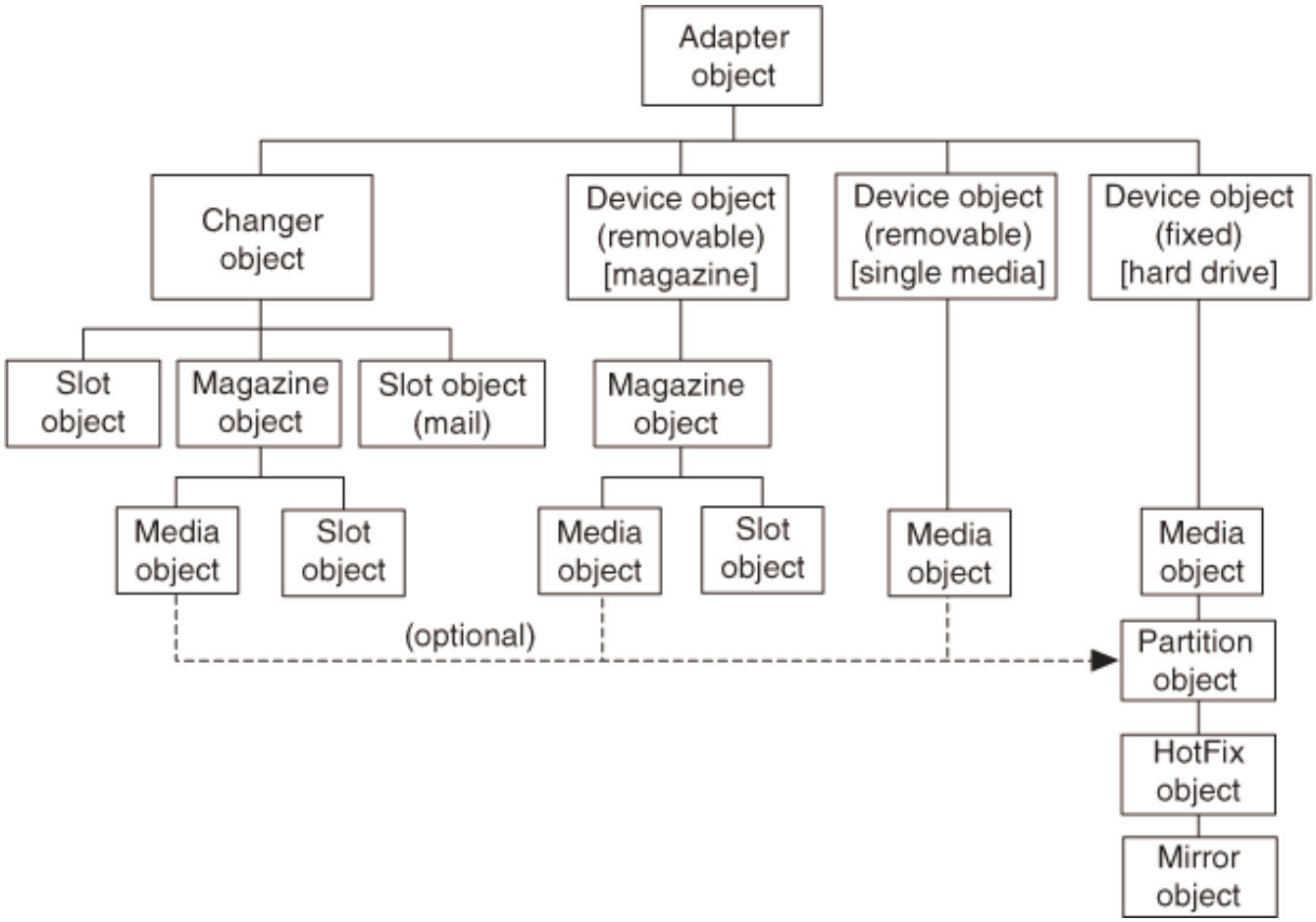
Media Manager Introduction

Media Manager Database Concepts

The Media Manager maintains a database of storage objects in memory. These objects are used to represent several types of devices.Novell® may in the future create and manage new classes of storage objects. Currently, the object IDs that Media Manager uses to represent these objects are dynamic (and reusable), but this may change in future implementations of the Media Manager.

Storage objects are linked together to form storage dependencies. A storage dependency exists when an object has no value unless physically or logically combined with another object. For example, a function call request to read a tape makes no sense unless the tape drive is activated and the tape inserted in the drive. So, in the Media Manager database, the tape drive's associated Device Object (Removable) is a "parent" of the tape's associated Media Object (Removable). Objects are said to be a child of an object on which it depends as shown in the following figure. An object's siblings are the group of objects having the same parent.

Figure 9. The Media Manager Database



Adapter Object

The Adapter Object represents a host adapter card that supports physical devices. The Media Manager adds an Adapter Object to the database when an adapter driver (for example, a Host Adapter Module in the NWPA architecture) registers with the Media Manager. An Adapter Object does not have any parents, but it may have Changer or Device Objects as children.

Changer Object

The Changer Object represents a random access media movement device in a media changer (or "jukebox"). In a standard configuration, a jukebox would also have one or more Device Objects representing the media drives contained in it. These drives would be identified separately in the database,

usually as siblings to the Changer Object. The Media Manager adds a Changer Object to the database when a changer device driver first registers with the Media Manager. A Changer Object is always a child of an Adapter Object. It may have Slot and Magazine Objects as children.

Device Objects

A Device Object is always the child of an Adapter Object. A Device Object may be a parent of Magazine Objects or Media Objects. The two types of Device Objects are Removable and Fixed.

Removable	The Device Object (Removable) is used to represent removable media devices, such as Magneto-Optical (MO) drives, CD-ROM drives, and tape drives. When a removable device driver first registers with the Media Manager, the Media Manager adds a Device Object (Removable) to the database.
Fixed	The Device Object (Fixed) is used to represent fixed (nonremovable) media devices, such as hard drives. When a fixed device driver first registers with the Media Manager, the Media Manager adds a Device Object to the database.

Slot Object

A Slot Object corresponds to a single media storage slot in a changer. Slot Objects are added to the database when a changer's device driver registers with the Media Manager. A Slot Object can be the child of either a Changer Object or a Magazine Object. A Slot Object (Mail) is a special type of Slot Object that represents the import/export elements in a media changer.

Magazine Object

The Magazine Object represents a set of media contained in a single cartridge or magazine. A CD-ROM magazine, for example, often contains 6 to 12 CD-ROM disks at once. Automated CD-ROM or MO jukeboxes may contain even larger numbers of disks. When a device driver reports a media magazine in a device, the Media Manager adds a Magazine Object to the database. A Magazine Object may stand alone or it may be the child of either a Removable Device or Changer Object. It may also be the parent of Slot or Media Objects. Each piece of media in a magazine would be a Media Object child of the Magazine Object.

Media Objects

Applications can label, unlabel, and identify Media Objects in the database independent of the physical medium. A Media Object may stand alone, or it

may be a child of a Device or Magazine Object. It may also be a parent to a Partition Object. The two types of Media Objects are Removable and Fixed.

Removable	A Media Object (Removable) represents physical pieces of removable media such as tapes, disk cartridges, MO disks or CD-ROMs.
Fixed	A Media Object (Fixed) corresponds to fixed (nonremovable) media on the system. Objects of this type can be created dynamically when detected in magazines, devices or changers, or they can be added to the database by an application's use of **MM_Create_Media_Object**.

Partition Object

NetWare® divides storage media into logical sections for ease of management. The Partition Object represents one of these logical sections of a physical media. Partition Objects can only be children of a Media Object and may be a parent to a Hot Fix™ Object.

Hot Fix Object

The Hot Fix Object represents the physical location of the Hot Fix area on a NetWare volume. Hot Fix Objects can only be children of a Partition Object and may be a parent to a Mirror Object.

Mirror Object

Disk mirroring is a fault tolerant feature that used to require its own interface to implement. Now, as an object in the Media Manager database, the Mirror Object is used to represent a physical device that will receive the same input as a primary storage device. Mirror Objects can only exist as a child of a Hot Fix Object.

Parent Topic:

Media Manager Introduction

Media Manager Queue Management

This section applies only to media devices, such as tape drives, which are sequentially accessed. Upon any error or exception condition (e.g., reaching End of Media, File Mark detected while reading, etc.), the device driver freezes the queue. This allows the application to decide whether to proceed

with the currently queued operations by unfreezing the queue thus allowing the queued requests to proceed normally, or to abort them. The first case might occur in an application which is restoring multiple sessions from a single media when one session might end with a File Mark but the application wants to continue reading the requests for the second session (which are beyond the File Mark.) In the second case the application might abort the pending request so the media doesn't go beyond the File Mark.

The frozen queue only affects I/O action codes (described in Storage Access I/O Action Codes (0x0020 - 0xFFFF)). Control action codes (described in Storage Access Control Action Codes (0x0000 - 0x001F)), such as Mount/Dismount, Lock/Unlock etc., can still be issued since they are issued one at a time on a priority basis (that is, bypassing frozen queues and pending I/Os). If the device has been deactivated because of a fatal error no further I/Os can be issued until the device is reactivated at which time the queue is automatically unfrozen.

In either case the queue **must** be unfrozen by the application. Even if the application is finished with the device, it should unfreeze the queue for future use by it or another application. The queue is unfrozen by issuing the I/O action code Reset Queue (action code number 0x25). This action code is described in Action Code: Reset Queue (0x0025).

Parent Topic:

Media Manager Introduction

Media Types

Used in GenericInfoDef

```
0x00   direct-access device (magnetic disk)
0x01   sequential-access device (magnetic tape)
0x02   printer device
0x03   processor device
0x04   write once device (some optical disks)
         /* Media Types (Continued) */
0x05   CD-ROM device
0x06   scanner device
0x07   optical memory device (some optical disks)
0x08   medium changer device (jukeboxes)
0x09   (Reserved)
0x0A-0x0B   defined by ASC IT8 (Graphic Arts Pre-Press)
0x0C-0x1E   Reserved
0x1F   unknown or no device type
0xFFCall    CDM_Inquiry( ) function (See NWPA Functional
 Specification) for every type of device
```

Parent Topic:

Function/Completion Code Definitions

Message Actions

```
#define LOCK_FUNCTIONS                0x0007
#define MOVE_FUNCTIONS                0x0008
#define STAMP_FUNCTIONS               0x0009
#define SCAN_FUNCTIONS                0x000A
#define CHECK_FUNCTIONS               0x000B
#define IDENTIFY_FUNCTIONS            0x000C
#define LOAD_MAGAZINE_FUNCTIONS       0x000D
#define CHANGER_INVENTORY_FUNCTIONS   0x000E
#define RAW_INSERT_FUNCTION           0x001B
#define RAW_CHANGER_FUNCTION          0x001C
#define RAW_MAGAZINE_FUNCTION         0x001D
#define RANDOM_READ                   0x0020
#define RANDOM_WRITE                  0x0021
#define RANDOM_WRITE_ONCE             0x0022
#define SEQUENTIAL_READ               0x0023
#define SEQUENTIAL_WRITE              0x0024
#define RESET_QUEUE                   0x0025
#define SINGLE_FILE_MARKS             0x0026
#define MULTIPLE_FILE_MARKS           0x0027
#define SINGLE_SET_MARKS              0x0028
#define MULTIPLE_SET_MARKS            0x0029
#define SPACE_DATA_BLOCKS             0x002A
#define LOCATE_DATA_BLOCKS            0x002B
#define PARTITION_SUPPORT             0x002C
#define SEQUENTIAL_SUPPORT            0x002D
#define DEVICE_GENERIC_IOCTL            0x003E
```

Parent Topic:

Function/Completion Code Definitions

Notification Functions

The Media Manager has the ability to alert an application when the device environment changes. The Media Manager can alert the application only if the Media Manager has the address of the application's custom-built notify function. First, the application must register the name and location of its own notify function with the Media Manager. As part of the registration process, the application provides the Media Manager with a list of environment changes the application is interested in. The Media Manager will then use this notification function to notify the application if a selected environment change occurs. Using this information, the application can then respond correctly to the change. The Notification Functions and the associated Application function are:

MM_Register_Notify_Routine

NotifyFunction (Application Function)

MM_Unregister_Notify_Routine

Parent Topic:

Media Manager: Guides

Notify Event Bits

Used in **MM_Register_Notify_Routine**.

```
#define NOTIFY_OBJECT_CREATION        0x0001
#define NOTIFY_OBJECT_DELETION        0x0002
#define NOTIFY_OBJECT_ACTIVATED       0x0004
#define NOTIFY_OBJECT_DEACTIVATED     0x0008
#define NOTIFY_OBJECT_RESERVATION     0x0010
#define NOTIFY_OBJECT_UNRESERVATION   0x0020
#define NOTIFY_OBJECT_PRECREATION     0x0040
#define NOTIFY_OBJECT_PREDELETION     0x0080
#define NOTIFY_OBJECT_ATTRIBUTE_CHANGE   0x0100
```

Parent Topic:

Function/Completion Code Definitions

Object Status Bits

Used in the DeviceInfoDef structure.

```
#define OBJECT_ACTIVATED          0x00000001
#define OBJECT_CREATED            0x00000002
#define OBJECT_ASSIGNABLE         0x00000004
#define OBJECT_ASSIGNED           0x00000008
#define OBJECT_RESERVED           0x00000010
#define OBJECT_BEING_IDENTIFIED   0x00000020
#define OBJECT_MAGAZINE_LOADED    0x00000040
#define OBJECT_FAILURE            0x00000080
#define OBJECT_REMOVABLE          0x00000100
#define OBJECT_READ_ONLY          0x00000200
#define OBJECT_INFO_VALID         0x00000400
#define OBJECT_IN_DEVICE          0x00010000
#define OBJECT_ACCEPTS_MAGAZINES  0x00020000
#define OBJECT_IS_IN_A_CHANGER    0x00040000
#define OBJECT_LOADABLE           0x00080000
#define OBJECT_DEVICE_LOCK        0x01000000
#define OBJECT_CHANGER_LOCK       0x02000000
#define OBJECT_REMIRRORING        0x04000000
#define OBJECT_SELECTED             0x08000000
```

Parent Topic:

Function/Completion Code Definitions

Object Support Functions

These functions are used to manage objects in the Media Manager database. Object Support Functions are:

MM_Create_Media_Object

MM_Delete_Media_Object

MM_Rename_Object

Parent Topic:

Media Manager: Guides

Object Types

Used by **MM_Find_Object_Type**.

```
#define ADAPTER_OBJECT           0
#define CHANGER_OBJECT           1
#define DEVICE_OBJECT            2
/* (reserved)                    3 */
#define MEDIA_OBJECT             4
#define PARTITION_OBJECT         5
#define SLOT_OBJECT              6
#define HOTFIX_OBJECT            7
#define MIRROR_OBJECT            8
#define PARITY_OBJECT            9
#define SEGMENT_OBJECT          10
#define VOLUME_OBJECT           11
#define CLONE_OBJECT            12
/* (reserved)                   13 */
#define MAGAZINE_OBJECT         14
#define VIRTUAL_DEVICE_OBJECT   15
#define UNKNOWN_OBJECT            0xFFFF
```

Parent Topic:

Function/Completion Code Definitions

Partition Management Functions

This set of functions provides a way for applications to create, delete and

manage Partition, Hot Fix, and Mirror Objects.

MM_CreateHotFix

MM_CreateMirror

MM_CreatePartition

MM_DeleteHotFix

MM_DeleteMirror

MM_DeletePartition

MM_DelPartitionFromMirrorGroup

MM_ForceMirrorGroupInSync

MM_InitializePartitionTable

MM_RemirrorGroup

MM_ReturnMirrorInfo

MM_ReturnPartitionTableInfo

Parent Topic:

Media Manager: Guides

Related Topics:

Using Media Manager Partition Functions

Reservation Functions

When several applications access the same pool of storage devices, contention is inevitable. Therefore, Media Manager is equipped with a suite of functions that allows applications to reserve an object for its own use. The reservation is done only when needed, and not for the duration of program execution. The Reservation functions and the associated Application functions are:

MM_Register_Application

ConsoleFunction (Application Function)

MM_Release_Object

MM_Release_Unload_Semaphore

MM_Reserve_Object

AlertFunction (Application Function)

MM_Set_Unload_Semaphore

MM_Unregister_Application

Parent Topic:

Media Manager: Guides

Reservation Modes

Used by **MM_Reserve_Object**

```
#define MODE_IO          0
#define MODE_CONTROL     1
#define MODE_RESERVED     7
```

Parent Topic:

Function/Completion Code Definitions

Resource Tag Allocation Signatures

```
#define MMApplicationSignature0x50424D4D
   /* 'PAMM'  (see MM_Register_Application)*/
#define MMNotifySignature0x4F4E4D4D
   /* 'ONMM'  (see MM_Register_Notify_Routine) */
#define MMIdentifySignature0x44494D4D
   /* 'DIMM'   (see MM_Register_Identification_Routines)*/
```

Parent Topic:

Function/Completion Code Definitions

Vendor Pass-Through Functions

This set of functions provide applications the ability to communicate directly with a device or an adapter. Possible uses include passing proprietary information to devices during hardware configuration (for example, during RAID system configuration), or receiving diagnostic information from a device or adapter during operation. These functions are described in the *NetWare Peripheral Architecture Functional Specification and Developer's Guide* (also available from Novell Labs™), since they are part of that architecture.

Parent Topic:

Media Manager: Guides

Media Manager: Functions

HJ_Media_Request

Prompts the operator at the console to insert or remove media from a device
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG HJ_Media_Request(
   struct *InsertRequestDef   minfo,
   LONG                       requestCode,
   LONG                       uniqueID);
```

Parameters

minfo

(IN) Pointer passed to the **ConsoleFunction** that the application registers via **MM_Register_Application**.

requestCode

(IN) Defined in the **ConsoleFunction** sample specification under **MM_Register_Application**.

uniqueID

(IN) Passed to and from the **ConsoleFunction** registered in **MM_Register_Application**.

Return Values

`HJ_ERROR`

Media count in the *InsertRequestDef* structure was either < 0 or > 1024, or the media number was not -1.

> **NOTE:** The maximum allowed media count limit of 1024 is currently set in the Media Manager, however, this will be changed in a future release. See Function/Completion Code Definitions for additional completion codes.

HJ_Media_Request_Ack

Acknowledges an outstanding media request
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG  HJ_Media_Request_Ack (
   struct InsertRequestDef   *minfo,
   LONG                      ackCode,
   LONG                      uniqueID)
```

Parameters

minfo

(IN) Pointer passed to the **ConsoleFunction** that the application registers via **MM_Register_Application**.

ackCode

(IN) HJ_ACK_MESSAGE HJ_NACK_MESSAGE

uniqueID

(IN) Passed to and from the **ConsoleFunction** registered in **MM_Register_Application**

Return Values

MM_OK

Function completed successfully

Remarks

The application's **ConsoleFunction** (see **MM_Register_Application**) calls this function to acknowledge an outstanding media request.

MM_Abort_Function

Attempts to abort a currently active request
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG    MM_Abort_Function(
   LONG    osRequestHandle)
```

Parameters

osRequestHandle

(IN) This parameter is the request handle filled in by **MM_Object_IO**. It identifies the request to abort.

Return Values

`MM_OK`

Function completed successfully

`MM_PARAMETER_ERROR`

osRequestHandle was set incorrectly.

Remarks

This is a nonblocking function. It attempts to abort a currently active request. The callback function will still be called even with an "abort" status.

MM_AddMirrorObjectToMirrorGroup

Adds an unreserved Mirror Object to a mirror group
Classification: 3.12, 4.*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_AddMirrorObjectToMirrorGroup(
   LONG   mirrorObjectID,
   LONG   mirrorGroupObjectID);
```

Parameters

mirrorObjectID

(IN) The Media Manager object ID of the **unreserved** Mirror Object to be added to the mirror group.

mirrorGroupObjectID

(IN) The Media Manager object ID of the **unreserved** mirror group to which the Mirror Object will be added.

Return Values

MM_OK

Function completed successfully.

MM_INVALID_OBJECT

Either *mirrorObjectID* or *mirrorGroupObjectID* is invalid (not in the database or not a Mirror Object) or the Mirror Objects they refer to do not have either a Hot Fix Object for a parent or a Partition Object for a grandparent.

MM_INVALID_MODE

The object was not active.

MM_RESERVATION_CONFLICT

The Mirror Objects selected were reserved. This function requires that the mirror objects be unreserved.

MM_PARAMETER_ERROR

One or more parameters passed is not what was expected.

MM_FAILURE

The function failed to add the Mirror Object to the group.

The function failed to add the Mirror Object to the group.

Remarks

This function adds an unreserved Mirror Object identified by the object ID *mirrorObjectID* to the mirror group identified by the object handle *mirrorGroupObjectID* to create a single Mirror Object. The old Mirror Object associated with the *mirrorObjectID* will be removed from the Media Manager's database and will no longer be accessible.

IMPORTANT: This means that any existing data associated with the Mirror Object *mirrorObjectID* will also be overwritten after this function is used. For details, please refer to Partition API Usage in this addendum.

NOTE: Between 1 and 8 mirror objects can be part of a mirror group.

MM_Check_For_Pending_Aborts

Checks for abort requests that have been issued, but not completed because the driver still has data to flush

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG   MM_Check_For_Pending_Aborts(void);
```

Return Values

`0`

No pending aborts; the driver can release memory associated with allocated requests.

`nonzero`

The pending number of aborts

MM_Create_Media_Object

Identifies function to add the newly identified objects to the Media Manager database

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>


LONG   MM_Create_Media_Object(
   LONG                  objectID,
   struct MediaInfoDef   *mediaInfo);
```

Parameters

objectID

(IN) This parameter is the internal handle passed to the Identification Function, identifying the device with the media.

Note: Use this function to add unidentified new media objects to the database by passing a -1 as the object ID. New objects could represent media in external storage cabinets that need to be inserted by console operators. See the Human Jukebox APIs)

mediaInfo

(IN) This parameter points to a MediaInfoDef structure containing information about the media.

Return Values

MM_OK

No pending aborts; the driver can release memory associated with allocated requests.

MM_MEMORY_ALLOCATION_ERROR

Memory allocation failed.

MM_PARAMETER_ERROR

objectID was not in the database or was not set to -1.

Remarks

This function is a nonblocking function. It checks for abort requests that

have been issued, but not completed because the driver still has data to flush.

MM_CreateHotFix

Creates a redirection area for a partition
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_CreateHotFix(
   LONG   partitionObjectHandle,
   LONG   dataSize,
   LONG   *newHotFixObjectID);
```

Parameters

partitionObjectHandle

(IN) The Media Manager object handle of the reserved Partition Object where the Hot Fix area will be created.

dataSize

(IN) The number of sectors to be contained in the usable data portion of the Hot Fix Object.

newHotFixObjectID

(OUT) A pointer to the Media Manager object ID of the Hot Fix object created by this function.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

partitionObjectHandle was invalid

`MM_INVALID_MODE`

The object was not active.

`MM_RESERVATION_CONFLICT`

The Partition Object was not reserved by the application prior to the request.

`MM_PARAMETER_ERROR`

One or more parameters passed is not what was expected.

`MM_FAILURE`

The function failed to create a Hot Fix Object .

Remarks

This function creates a redirection area for the partition selected. If one already exists, it will first be deleted. A Hot Fix object is also created in the Media Manager database by this function.

NOTE: 1. Currently the maximum size for the redirection area is 120 megabytes. Any request for a redirection size greater than that amount will be truncated at 120 megabytes. However, the size of the data portion of the partition will contain the requested size.

2. The data size of a currently defined Hot Fix object can be found by reading the *capacity* field in the Media Manager structure GenericInfoDef for the Hot Fix object.

MM_CreateMirror

Creates a Mirror area on the selected Partition Object
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_CreateMirror(
   LONG   partitionObjectHandle,
   LONG   *newMirrorObjectID);
```

Parameters

partitionObjectHandle

(IN) The Media Manager object handle for the reserved Partition Object.

newMirrorObjectID

(IN) A pointer to the Media Manager object ID for the newly created Mirror Object.

Return Values

MM_OK

Function completed successfully.

MM_INVALID_OBJECT

partitionObjectHandle was invalid (not in the database) or the child of the Partition Object selected was not a Hot Fix Object

MM_INVALID_MODE

The object was not reserved in the proper mode (i.e., I/O vs. Control) for this operation.

MM_RESERVATION_CONFLICT

The Partition Object was not reserved by the application prior to the request.

MM_PARAMETER_ERROR

One or more parameters passed is not what was expected.

MM_FAILURE

The function failed to create a Mirror Object.

Remarks

This function creates a Mirror area on the selected Partition Object. A Mirror Object is also added to the Media Manager database.

MM_CreatePartition

Finds the first available physical partition on a disk and allocates it to NetWare

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_MM_CreatePartition(
   LONG   deviceObjectHandle,
   LONG   partitionType,
   LONG   startingSector,
   LONG   numberOfSectors,
   LONG   *newPartitionObjectID);
```

Parameters

Note: The parameters *areaStart* and *alignment* listed below are defined in the function **MM_ReturnPartitionTableInfo** and are returned from that function

deviceObjectHandle

(IN) The Media Manager object handle of the reserved Device Object.

partitionType

(IN) The identification type for the partition to be created. NETWARE_PARTITION_386 is the only type presently supported by this function.

startingSector

(IN) The beginning absolute sector number for the new logical partition. If this is the first partition it must be equal to areaStart. If this is not the first partition this must be a multiple of alignment. See Note above.

numberOfSectors

(IN) The number of sectors to be assigned to the new logical partition. This must be calculated so that the ending sector (the *startingSector* plus the *numberOfSectors*) is a multiple of alignment. See Note above

newPartitionObjectID

(OUT) The pointer to the Media Manager object ID of the newly created Partition Object.

Return Values

MM_OK

Function completed successfully.

MM_INVALID_OBJECT

The object was not active.

MM_INVALID_MODE

The object was not reserved in the proper mode (i.e., I/O vs. Control) for this operation.

MM_RESERVATION_CONFLICT

The selected Device Object wWas not reserved by the application prior to the request

MM_PARAMETER_ERROR

partitionType was not set to NETWARE_PARTITION_386 or one or more parameters passed is not what was expected.

MM_FAILURE

The function failed to create a partition.

Remarks

This function finds the first available physical partition on a disk and allocates it to NetWare using NetWare's assigned number. A Partition Object is added to the Media Manager database as a result of this function being called.

NOTE: The NetWare file system does not currently support more than one NetWare partition at a time per physical device.

MM_Delete_Media_Object

Deletes a media or magazine object
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Delete_Media_Object(
   LONG   objectNumber);
```

Parameters

objectNumber

(IN) Specifies the media object to be deleted.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

objectNumber is attached to something (i.e., media is in a device).

`MM_PARAMETER_ERROR`

objectID was not in the database or was not set to -1.

Remarks

This function deletes a media or magazine object. It can only delete media or magazine objects that are not attached. If a magazine object is specified, all media objects within the magazine are also deleted.

MM_DeleteHotFix

Removes the Hot Fix information from a Partition Object
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_DeleteHotFix(
   LONG   hotFixObjectID);
```

Parameters

hotFixObjectID

(IN) The Media Manager object ID of the Hot Fix object to be deleted. You must first reserve the obect before deleting.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

hotFixObjectID was invalid (not in the database) or the Hot Fix Object's parent was not a Partition Object

`MM_RESERVATION_CONFLICT`

The Hot Fix Object was already reserved at the time of the call.

`MM_PARAMETER_ERROR`

One or more parameters passed is not what was expected.

`MM_FAILURE`

The function failed to delete the Hot Fix area. For this function, this could be caused by the Hot Fix object still having child objects. All child objects of the Hot Fix object must be deleted prior to deleting the Hot Fix object.

Remarks

This function removes the Hot Fix information from a Partition Object. The related Hot Fix Object is also deleted from the Media Manager object database.

MM_DeleteMirror

Removes the mirror information from a Partition Object
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_DeleteMirror(
   LONG   mirrorObjectID);
```

Parameters

mirrorObjectID

(IN) The Media Manager object ID for the unreserved Mirror Object to be deleted.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

mirrorObjectID was invalid (not in the database) or the Mirror Object selected did not have either a Hot Fix Object for a parent or a Partition Object for a grandparent.

`MM_RESERVATION_CONFLICT`

The Mirror Object was already reserved at the time of the call.

`MM_FAILURE`

The function failed to delete the Mirror Object.

Remarks

This function removes the mirror information from a Partition Object. The related Mirror Object is also deleted from the Media Manager object database.

MM_DeletePartition

Deletes the NetWare partition on a device object
Classification: 3.12, 4,x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_DeletePartition(
   LONG   partitionObjectID,
   LONG   partitionType,
   LONG   startingSector,
   LONG   numberOfSectors);
```

Parameters

partitionObjectID

(IN) The Media Manager object handle of an unreserved Partition Object to be deleted.

partitionType

(IN) This is currently not used and must be set to -1.

startingSector

(IN) The beginning absolute sector number for the logical partition to be deleted.

numberOfSectors

(IN) The number of sectors in the partition object to be deleted. **Note:**If either the *startingSector* or *numberOfSectors* parameters is incorrect for the partition requested, this function will fail to delete the partition. Since this function is destructive, these parameters are used to verifiy that the partition requested matches the partition that is on the media before deleting the partition.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

partitionObjectID was invalid.

`MM_RESERVATION_CONFLICT`

The Partition Object was reserved prior to the request.

`MM_PARAMETER_ERROR`

partitionType was not set to -1 or one or more parameters passed is not what was expected.

`MM_FAILURE`

The function failed to delete the partition. For this function, this could be caused by the Partition object still having child objects. All child objects of this partition must be deleted prior to deleting the Partition object.

Remarks

This function will delete the NetWare partition on the device object specified by zeroing out the partition entry. It will also delete the related Partition Object from the Media Manager database.

MM_DelPartitionFromMirrorGroup

Removes a selected mirrored partition from the mirror group it is currently in

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_DelPartitionFromMirrorGroup(
   LONG   partitionObjectID);
```

Parameters

partitionObjectID

(IN) The Media Manager object ID of the **unreserved** Partition Object that is to be removed from its current mirror group.

partitionType

(IN) This is currently not used and must be set to -1

startingSector

(IN) The beginning absolute sector number for the logical partition to be deleted.

numberOfSectors

(IN) The number of sectors in the partition object to be deleted. **Note:** If either the *startingSector* or *numberOfSectors* parameters is incorrect for the partition requested, this function will fail to delete the partition. Since this function is destructive, these parameters are used to verifiy that the partition requested matches the partition that is on the media before deleting the partition.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

partitionObjectID was invalid.

`MM_RESERVATION_CONFLICT`

The Partition Object was reserved prior to the request.

`MM_PARAMETER_ERROR`

partitionType was not set to -1 or one or more parameters passed is not what was expected.

`MM_FAILURE`

The function failed to remove the Partition Object from its current mirror group.

Remarks

This function removes a selected mirrored partition from the mirror group it is currently in. A new out of sync mirror object will be created for the partition. This new mirror object may or may not have a different Object ID than it did before it was added to the mirror group. The Partition Objects that make up a mirror group can be determined by using the information returned by **MM_Return_Object_MappingInfo**. See the functional specification for details of that function.

IMPORTANT: An application must not perform this function on a partition which is part of a mirror group that is not currenly in sync unless it is sure that the partition to be removed contains the "out of sync" data.

MM_Find_Object_Type

Searches the database for specific object types
Classification: 3.12, 4,x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Find_Object_Type
   LONG   type,
   LONG   *nextIndicator);
```

Parameters

type

(IN) This is the class of object to be located in the database. See Function/Completion Code Definitions for the list of valid object types.

nextIndicator

(OUT) This is used to traverse through the Media Manager database. It contains an initial value of -1. During the search, nextIndicator contains the value of the last object ID searched for (see type above).

Return Values

MM_OK

Function completed successfully.

MM_OBJECT_NOT_FOUND

No type class objects were found in the database.

MM_PARAMETER_ERROR

objectID was not in the database or was not set to -1.

Remarks

This function allows applications to search the database for specific object types such as adapters, changers, media, partitions, etc. When **MM_Reserve_Object** is called to reserve an object, then **MM_Reserve_Object** is passed the object ID received by calling this function.

MM_ForceMirrorGroupInSync

Changes the status of a mirror group to synchronized
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_ForceMirrorGroupInSync(
   LONG   mirrorGroupObjectID);
```

Parameters

mirrorGroupObjectID

(IN) The Media Manager object ID of the mirror group to be declared synchronized.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

mirrorGroupObjectID was invalid (not in the database or not a Mirror Object) or it did not have either a Hot Fix Object as a parent or a Partition Object for a grandparent.

`MM_FAILURE`

The function failed to declare the mirror group synchronized.

Remarks

This function changes the status of a mirror group to synchronized. Currently, this function only affects the first two mirrored partitions in a mirror group, even though the group may have up to eight partitions in it.

NOTE: This function DOES NOT synchronize a mirrored partition. This function is used when a mirror group is created and is known to be empty to prevent the synchronizing process from mirroring any random data on a new volume. If an existing mirrored partition needs to be synchronized, use the **MM_RemirrorGroup** function. For a detailed discussion of the safe use of this function, see Partition API Usage in this addendum.

MM_InitializePartitionTable

Clears the partition table and makes it ready for its first partition to be defined

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_InitializePartitionTable(
   LONG   deviceObjectHandle);
```

Parameters

deviceObjectHandle

(IN) The Media Manager object handle of a reserved Device Object.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

deviceObjectHandle was invalid

`MM_INVALID_MODE`

The object was not active.

`MM_RESERVATION_CONFLICT`

The Device Object was not reserved by the application prior to the request.

`MM_FAILURE`

The function failed to initialize the partition table.

Remarks

This function clears the partition table and makes it ready for its first partition to be defined. (In the IBM partitioning scheme this will NULL sector 0 of the media, leaving the partition signature [0xAA55]).

NOTE: This function will deactivate Partition Objects which are child objects of the device. It will also remove the associated Partition Objects from the Media Manager database.

MM_Object_Blocking_IO

Issues I/O requests to an object
Classification: 3.12, 4,x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Object_Blocking_IO(
   LONG   *returnParameter,
   LONG   osReserveHandle,
   LONG   actionCode,
   LONG   parameter1,
   LONG   parameter2,
   LONG   parameter3,
   LONG   bufferLength,
   void   *buffer);
```

Parameters

returnParameter

(IN) This parameter points to a 4-byte location where returned information is copied (only if the return was successful).

osReserveHandle

(IN) This parameter is the *objectHandle* returned via MM_Reserve_Object. It specifies the target object for the I/O request.

actionCode

(IN) This parameter indicates which Control or I/O action code is to be performed (see Chapter 3 or Chapter 4 respectively).

parameters 1,2,3

(IN) The meaning of these parameters depends upon the action code specified (see Chapter 3, Chapter 4).

bufferLength

(IN) This parameter specifies the length of the buffer.

buffer

(IN) This parameter points to data or control buffers passed to the device. The buffer size is specified by bufferLength.

Return Values

`MESSAGE_PROCESSED`

Function completed successfully.

`MESSAGE_DATA_MISSING`

`MESSAGE_POSTPONE`

The operation failed because of memory allocation errors; you should retry the operation later

`MESSAGE_ABORTED`

`MESSAGE_INVALID_PARAMETERS`

osReserveHandle was not found in the database.

`MESSAGE_OBJECT_NOT_ACTIVE`

`MESSAGE_INVALID_OBJECT`

`MESSAGE_FUNCTION_NOT_SUPPORTED`

`MESSAGE_INVALID_MODE`

The mode for this object was not set to IO or Control Mode when it was reserved.

`MESSAGE_ABORTED_CLEAN`

Remarks

A blocking function. It issues I/O requests to the object specified by the object handle and returns when the action completes

MM_Object_IO

Issues I/O requests to an object
Classification: 3.12, 4,x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Object_IO (
   LONG   *osRequestHandle
   LONG   requestHandle,
   LONG   osReserveHandle,
   LONG   actionCode,
   LONG   parameter1,
   LONG   parameter2,
   LONG   parameter3,
   LONG   bufferLength,
   void   *buffer,
   void   (*callBackFunction));
```

Parameters

osRequestHandle

(OUT) Points to a 4-byte location where the Media Manager will copy the request handle associated with this request.

requestHandle

(IN) The meaning of this parameter is specific to the application. It is passed when **MM_Object_IO** calls **CallBackFunction**.

osReserveHandle

(IN) The reserve handle returned via **MM_Reserve_Object**. It specifies the target object for the I/O request.

actionCode

(IN) Indicates which Control or I/O action code is to be performed.

parameters 1,2,3

(IN) The meaning of these parameters depends upon the action code specified (see Chapter 3, Chapter 4).

bufferLength

(IN) Specifies the length of the buffer.

buffer

(IN and OUT) Points to data or control buffers passed to the device.

The buffer size is specified by *bufferLength*.

callBackFunction

(IN) Points to a call back function that is called by the Media Manager at interrupt time to post final completion of the I/0 function.

```
#include <nwmediam.h>

void  CallBackFunction(
   LONG   osRequestHandle,
   LONG   requestHandle,
   LONG   returnParameter,
   LONG   completionCode)
```

CallBackFunction This is an interrupt time function passed to the Media Manager when MM_Object_IO( ) is called. It is called by the Media Manager when an I/O request is completed. The requestHandle contains an application specified value that is normally used to identify the request. The meaning of the completion code depends upon the class of I/O request issued.

osRequestHandle

(IN) The OS request handle returned when **MM_Object_IO** was called.

requestHandle

(IN) The application request handle passed to the Media Manager when **MM_Object_IO** was called. Typically, applications use it to identify a request.

returnParameter

(IN) A 4-byte return value for the function (where applicable).

completionCode

(IN) Final Completion Code (received by application from device drivers) see Function/Completion Code Definitions.

Return Values

Initial Completion Codes (received by the application from the Media Manager): `MESSAGE_PROCESSED`

Function completed successfully.

`MESSAGE_POSTPONE`

The operation failed because of memory allocation errors; you should retry the operation later

`MESSAGE_INVALID_PARAMETERS`

osReserveHandle was not found in the database.

`MESSAGE_INVALID_MODE`

The mode for this object was not set to IO or Control Mode when it

was reserved.

Remarks

is a process nonblocking function that cannot be called at interrupt time. It issues I/O requests to the object specified by the *osReserveHandle*. It returns immediately with an intermediate completion code. If it is not successful (nonzero value returned), the **CallBackFunction** is NOT called. If successful (zero value), the final completion code will be posted when **CallBackFunction** is called. **Note: CallBackFunction** may be called prior to the return of **MM_Object_IO**. Applications can call **MM_Abort_Function** to abort the request issued by **MM_Object_IO** before **MM_Object_IO** calls **CallBackFunction** to post completion.

MM_Register_Application

Registers an application with theMedia Manager
Classification: 3.12, 4,x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG  MM_Register_Application (
   LONG   *osApplicationHandle,
   LONG   applicationHandle,
   BYTE   *name,
   LONG   (*newAssignmentFunction),
   LONG   (*consoleFunction),
   LONG   resourceTag);
```

Parameters

osApplicationHandle

(OUT) This parameter points to a four-byte location where the Media Manager copies the OS application handle if the registration is successful. This is used to reserve resources.

applicationHandle

(IN) The value of this parameter is specific to the application.

name

(IN) This parameter is used to register the name of the application. It is a 64-byte (max) length-preceded ASCII string (First byte holds the length and the next 63 bytes hold the name). The function will fail with a parameter error unless this field holds a length-preceded string.

newAssignmentFunction

(IN) Is currently zero (0).

consoleFunction

(IN)

```
#include <nwmediam.h>

LONG  ConsoleFunction(
   struct MediaRequestDef   *min
   LONG                     uniqueID,
   LONG                     functionCode)
```

ConsoleFunction This optional function is called by the OS to handle all

physical media movement operations. The **ConsoleFunction** is registered with the OS through **MM_Register_Application**. **Note:** The structure MediaRequestDef applies to magazines and changers.

minfo

(IN) Points to the MediaRequestDef structure.

uniqueID

(IN) This parameter is used to identify an outstanding request. It is passed back from **HJ_Media_Request_Ack**.

functionCode

(IN) This has two types of values returned:

1) Messages that can only be returned when there is a new request: `HJ_INSERT_MESSAGE` and `HJ_EJECT_MESSAGE`

2) Messages returned when there is an existing request: `HJ_ACK_MESSAGE` and `HJ_NACK_MESSAGE`

Completion Codes:

Screen Not Active = 0 can switch to default console screen

Screen Active = -1 (non zero); do not switch to default console screen

resourcetag

(IN) Resource tag obtained by calling the OS function **AllocateResourceTag** with **MMApplicationsSignature** (0x50424D4D) as the signature parameter.

Return Values

`MM_OK`

Function completed successfully.

`MM_PARAMETER_ERROR`

The resource tag type identified by *resourceTag* was not set to `MMApplicationsSignature` and/or name[0] was greater than 64.

MM_Register_Identification_Routines

MM_Register_Identification_Routines registers functions that write label information on storage media with the Media Manager

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG   MM_Register_Identification_Routines(
   LONG   *osIdentifyHandle,
   LONG   identifyHandle,
   LONG   (*identifyFunction),
   LONG   (*unlabelFunction),
   LONG   (*labelFunction),
   LONG   identificationType,
   BYTE   *identifierName,
   LONG   resourceTag);
```

Parameters

osIdentifyHandle

(OUT) This parameter is the osIdentifyHandle returned by **MM_Register_Notifiy_Routine**.

identifyHandle

(IN) This parameter is the Novell assigned media identification type used to distinguish between identification functions.

identifyFunction

(IN) This parameter points to a function that can be called by the Media Manager to identify media on the system.

```
#include <nwmediam.h>

LONG  IdentifyFunction(
   LONG   objectID)
```

IdentifyFunction Identifies mounted media for the Media Manager. The application's **IdentifyFunction** should use the Media Manager access function **MM_Special_Object_Blocking_IO** to read from the media. If the **IdentifyFunction** succeeds in reading from the media, it should then call the Media Manger support function **MM_Create_Media_Object** to register the media with the OS.

objectID

(IN) The object ID of the mounted device containing the identified media

`0`

The function identified the media successfully.

`nonzero`

The function could not identify the media.

unlabelFunction

This parameter points to an optional function that can be called by the Media Manager to remove identification information from a piece of media. Enter a "0" in this field if your application does not support an unlabel operation.

```
#include <nwmediam.h>


   LONG    UnlabelFunction(
   LONG    objectID)
```

UnlabelFunction This function removes identification information from the media. Use the function MM_Special_Object_Blocking_IO to write to the media.

objectID

(IN) The object ID of the mounted media containing the media being unlabeled.

`0`

Function completed successfully.

`nonzero`

Function did not complete successfully.

labelFunction

This parameter points to an optional function that can be called by the Media Manager to label a piece of media with identification information. Enter a "0" in this field if your application does not support a label operation.

```
#include <nwmediam.h>

LONG LabelFunction(
   LONG               objectID,
   struct mediaInfo   *mediaInfo
   )
```

LabelFunction This function labels media with specific identification

information. The application's **LabelFunction** should use the Media Manager access function **MM_Special_Object_Blocking_IO** to read from the media.If the LabelFunction succeeds in reading from the media, it should call the Media Manager support function **MM_Create_Media_Object** to register the media with the OS.

objectID

(IN) The object ID of the mounted media that contains the media being identified.

mediaInfo

(IN) Points to a structure containing information about the media.

`0`

Function completed successfully.

`nonzero`

Function did not complete successfully.

identificationType

(IN) Novell assigned identification type.

identifierName

(IN) 64-byte (max) length preceded ASCII string.

resourceTag

(IN) The resource tag is obtained by calling the OS function **AllocateResourceTag** with `MMIdentifySignature` (0x44494D4D) as the Media Manager resource signature.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_RESOURCETAG`

The resource tag identified by *resourceTag* was not `MMIdentifySignature`.

`MM_PARAMETER_ERROR`

identifierName[0] was greater than 64

Remarks

Applications may provide functions that write label information on storage media. **MM_Register_Identification_Routines** registers those functions with the Media Manager. Specifications for each function are given here, along with a description of the corresponding parameters.

MM_Register_Notify_Routine

Registers a notification function with the Media Manager
Classification: 3.12, 4.x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG   MM_Register_Notify_Routine(
   LONG   *osNotifyHandle,
   LONG   notifyHandle,
   LONG   (*notifyFunction),
   LONG   objectType,
   LONG   eventMask,
   LONG   resourceTag);
```

Parameters

osNotifyHandle

(OUT) 4-byte pointer to where the OS copies the notifyHandle.

notifyHandle

(IN) 4-byte value defined by the application to identify this notify function.

notifyFunction

(IN) Specifies a blocking function that the Media Manager calls when specified objects are added or removed from the database.

```
#include <nwmediam.h>

void   NotifyFunction(
   LONG   osNotifyHandle,
   LONG   notifyHandle,
   LONG   objectID,
   LONG   objectType,
   LONG   eventType)
```

osNotifyHandle

(IN) The *osNotifyHandle* returned by **MM_Register_Notify_Routine**.

notifyHandle

(IN) The notifyHandle passed to the Media Manager when **MM_Register_Notify_Routine** was called. It is normally used to identify the object with the application.

objectID

(IN) The ID of the object generating the notification. It is a 4-byte hexadecimal number that identifies the object in the Media Manager database.

objectType

(IN) The objectType of the object generating the notification (See Function/Completion Code Definitions).

eventType

(IN) Indicates the type of event that generated the notification. (See Notify Event Bits in Function/Completion Code Definitions.)

objectType

(IN) Indicates the object Type of object to notify about.

eventMask

(IN) Indicates the type of event that will trigger notification.

resourceTag

(IN) Resource tag obtained by calling the OS function **AllocateResourceTag** using `MMNotifySignature` (0x4F4E4D4D) as the signature parameter.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_RESOURCETAG`

The resource tag identified by *resourceTag* was not `MMIdentifySignature`.

`MM_MEMORY_ALLOCATION_ERROR`

Memory allocation failed.

Remarks

Registers a notification function with the Media Manager. The application's notification function will be called when alerts are sent from the system.

NOTE: These notifications will be started in order (i.e., creation, then activation, etc.), but subsequent notifications will not wait for previous notifications to complete before starting (e.g., activation may be called before creation has completed, or reservation may be called before either creation or activation is completed. See the note on the Mount/Dismount function).

MM_Release_Object

Releases a "lock" on an object that was previously reserved by **MM_Reserve_Object**

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG   MM_Release_Object(
   LONG   osReserveHandle,
   LONG   reserveHandle);
```

Parameters

osReserveHandle

The OS reserve handle returned by **MM_Reserve_Object**.

reserveHandle

The reserve handle supplied by the application in **MM_Reserve_Object**.

Return Values

`MM_OK`

Function completed successfully.

`MM_PARAMETER_ERROR`

osReserveHandle was invalid

`MM_MEMORY_ALLOCATION_ERROR`

Memory allocation failed.

Remarks

The function releases a "lock" on an object that was previously reserved by **MM_Reserve_Object**. Objects can be automatically released on certain alerts or a driver unload operation.

MM_Release_Unload_Semaphore

Releases the semaphore on a database object after the object has been successfully removed

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG  MM_Release_Unload_Semaphore(
   LONG   currentInstance);
```

Parameters

currentInstance

Uniquely identifies the "release" semaphore command with the correct "set" semaphore command

Return Values

`0`

Function completed successfully.

`-1`

Error; invalid current instance (not in correct context)

Remarks

Releases the semaphore on a database object after the object has been successfully removed. It is called from the application's Alert function. **MM_Set_Unload_Semaphore** describes this process.

MM_Rename_Object

Changes an object's name in the database
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG   MM_Rename_Object(
   LONG   objectID,
   BYTE   *name);
```

Parameters

objectID

The ID of the object to be renamed. It is a 4-byte hexadecimal number that identifies the object in the Media Manager database.

name

The 64-byte (max) length-preceded ASCII string that holds the identifying name of the database object. (Byte [0] holds the length and bytes [1] to [63] hold the ASCII string).

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

objectID was not found in the database.

`MM_MEMORY_ALLOCATION_ERROR`

Memory allocation failed.

`MM_PARAMETER_ERROR`

name[0] was greater than 64

Remarks

changes an object's name in the database. It can only change the name that exists as an ASCII string in the name field of the `GenericInfoDef` structure. This function does not change the label field of the `MediaInfoDef` structure. The label field can only be changed by **MM_Create_Media_Object** or by an identification function registered

File Service Group

via **MM_Register_Identification_Routines**

MM_Reserve_Object

Reserves an object for exclusive use by the application
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG   MM_Reserve_Object(
   LONG   *osReserveHandle,
   LONG   reserveHandle,
   LONG   objectID,
   LONG   reserveMode,
   LONG   osApplicationHandle,
   LONG   (*AlertFunction));
```

Parameters

osReserveHandle

Points to the handle used as the*objectID* in all functions where the object needs to be reserved before the function can operate, such as **MM_Object_IO** and **MM_Object_Blocking_IO**.

reserveHandle

The value of this parameter is specified by the application and is passed as a parameter when **AlertFunction** is called.

objectID

The ID of the object to be reserved. It is a 4-byte hexadecimal number that identifies the object in the Media Manager database.

reserveMode

Selects I/O, control or reserved functions: `MODE_IO` `MODE_CONTROL` `MODE_RESERVED`

osApplicationHandle

Handle returned from **MM_Register_Application**.

AlertFunction

Points to a nonblocking interrupt time function that Media Manager can call when changes to object status occur (see sample **AlertFunction** below).

```
#include <nwmediam.h>

void  AlertFunction(
```

```
LONG   osReserveHandle,
LONG   reserveHandle,
LONG   alertType,
LONG   alertReason)
```

AlertFunction This is an interrupt time function passed to the Media Manager when **MM_Reserve_Object** is called. It is called by the Media Manager as a nonblocking call to inform the application of alerts related to the reserved object. The parameters *alertType* and *alertReason* indicate what kind of event has occurred on the object (see Function/Completion Code Definitions).

osReserveHandle

The OS reserve handle returned by **MM_Reserve_Object**.

reserveHandle

The reserve handle passed to the Media Manager when **MM_Reserve_Object** was called. It is normally used to identify the object with the application.

alertType

Contains a 4-byte alert code that indicates the type of alert. See Alert Types for this list.

alertReason

Contains a 4-byte code that explains the reason for the alert message. See Alert Reasons for the list of alert reasons.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

objectID was not found in the database.

`MM_MEMORY_ALLOCATION_ERROR`

Memory allocation failed.

`MM_INVALID_APPLICATION`

osApplicationHandle was invalid.

`MM_INVALID_MODE`

Mode selected was not IO mode, control mode or reserved mode or object was not activated.

`MM_RESERVATION_CONFLICT`

Object is already reserved.

Remarks

This function reserves an object for exclusive use by the application. The object must be reserved in a mode specified by the *reserveMode* parameter. Only action codes matching the reserved mode will be allowed to be performed. If another mode is needed by an application, it may re-reserve the object in the desired mode. It is not necessary for the application to call **MM_Release_Object** before reserving the object in the new mode. When the application is finished with an object, however, a call should be made to **MM_Release_Object** . This will release the object so that other applications may use it.

MM_RemirrorGroup

Sets the mirror group status to unsynchronized
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_RemirrorGroup(
   LONG   mirrorGroupObjectID,
   LONG   modeFlags);
```

Parameters

mirrorGroupObjectID

(IN) The Media Manager object ID of the mirror group to be synchronized.

modeFlags

(IN) Must be set to 0. In the future, this parameter will be used to select different synchronization modes.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

mirrorGroupObjectID was invalid (not in the database or not a Mirror Object) or it did not have either a Hot Fix Object as a parent or a Partition Object for a grandparent.

`MM_PARAMETER ERROR`

modeFlags was not set to 0 or mirrorGroupObjectID was invalid.

`MM_FAILURE`

The function failed to declare the mirror group unsynchronized.

Remarks

This function sets the mirror group status to unsynchronized, which starts the synchronization process. Currently, this function only affects the first two mirrored partitions in a mirror group, even though the group may have up to eight partitions in it.

MM_Return_Object_Attribute

Returns a specific attribute of an object
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Return_Object_Attribute(
   LONG   objectID,
   LONG   attributeID,
   LONG   length,
   void   *info);
```

Parameters

objectID

This is the ID of the object whose attribute will be returned. The ID is a 4-byte hexadecimal number that identifies the object in the Media Manager database.

attributeID

This specifies the ID of the attribute to be returned. The attribute IDs are listed in Attribute IDs.

length

This specifies the size of the buffer in which object attribute will be returned.

info

This points to a structure where object specific information (such as the object's attributes) will be copied. The format of this structure depends upon the object class and application type (see Media Manager: Structures).

Return Values

`MM_OK`

Function completed successfully.

`MM_PARAMETER_ERROR`

objectID was not found in database.

Remarks

Returns a specific attribute of an object. The attributes of an object depend upon the object type.

MM_Return_Object_Generic_Info

Returns generic information about an object
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG   MM_Return_Object_Generic_Info(
   LONG                    objectID,
   struct GenericInfoDef   *info);
```

Parameters

objectID

This is the ID of the object for which generic information will be returned. It is a 4-byte hexadecimal number that identifies the object in the Media Manager database.

info

This is a pointer to a GenericInfoDef structure that is passed to the application. This structure receives generic information about a database object.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

objectID was not found in the database.

Remarks

This function returns generic information about an object. Generic information includes a label, an object identification type (see "Database Concepts" in Chapter 1), and an identifying time stamp. Use **MM_Return_Object_Table_Size** to find out the maximum number of objects in the database.

MM_Return_Object_Mapping_Info

Returns the object IDs of parent, sibling, and child objects related to the object associated with an object

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Return_Object_Mapping_Info(
   LONG   objectID,
   LONG   mappingInfoLength,
   LONG  *mappingInfo);
```

Parameters

objectID

This is the ID of the object about which mapping information will be returned. It is a 4-byte hexadecimal number that identifies the object in the Media Manager database.

mappingInfoLength

This indicates the size in bytes of the buffer where the mapping information is to be copied. The size must be calculated by adding one LONG for each parent, child, and sibling to the minimum size of struct `MappingInfo` (3 LONGs). The number of parents, siblings, and children is reported by the `GenericInfoDef` structure which is returned from a call to **MM_Return_Object_Generic_Info**.

mappingInfo

This points to a MappingInfo structure that contains the mapping information.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

objectID was not found in the database.

`MM_PARAMETER_ERROR`

The *mappingInfoLength* buffer is not large enough for the requested information.

Remarks

Returns the object IDs of parent, sibling, and child objects related to the object associated with the objectID parameter. Call this function to find out device dependencies of certain objects in the database.

MM_Return_Object_Specific_Info

Returns additional information about an object
Classification: 3.12, 4,x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Return_Object_Specific_Info (
   LONG   objectID,
   LONG   infoLength,
   void   *info);
```

Parameters

objectID

This is the ID of the object about which additional information will be returned. It is a 4-byte hexadecimal number that identifies the object in the Media Manager database.

infoLength

This indicates the size of the buffer where the information is to be copied.

info

This points to a structure where object specific information will be copied. The format of this structure depends upon the object class and application type (see Media Manager: Structures).

Return Values

MM_OK

Function completed successfully.

MM_INVALID_OBJECT

objectID was not found in the database.

MM_PARAMETER_ERROR

buffer of size infoLength is not large enough for the requested information.

Remarks

This function returns additional information about an object. The format

and size of the information depend upon the object's type. Before using this call, a call to **MM_Return_Object_Generic_Info** is required to determine the amount of memory that needs to be allocated.

MM_Return_Object_Table_Size

Returns the maximum number of objects in the current database
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Return_Object_Table_Size (void);
```

Return Values

Returns a number reporting the number of objects in the current database.

MM_Return_Objects_Attributes

This function returns an object's attributes for an object identified by passing the *objectID* and the *attributeID*. This function then returns an `AttributeInfoDef` structure containing information about an object's attributes.

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Return_Objects_Attributes(
   LONG                       objectID,
   LONG                       attributeID,
   struct AttributeInfoDef    *info);
```

Parameters

objectID

This is the ID of the object whose specific attributes will be returned. The objectID is a 4-byte hexadecimal number that identifies the object in the Media Manager database.

attributeID

This is the ID of the attribute for which specific attribute information will be returned. The attribute ID is the first four bytes (swapped) of the attribute name. (See Function/Completion Code Definitions.)

info

This points to an AttributeInfoDef structure containing information about an object's attributes.

Return Values

`MM_OK`

Function completed successfully.

`MM_PARAMETER_ERROR`

buffer of size infoLength is not large enough for the requested information.

Remarks

This function returns the maximum number of objects in the current

database. The state of the database is dynamic. Use this function to "bounds check" the database whenever the number of objects in the database is needed to be known.

MM_ReturnMirrorInfo

Returns mirror information for a selected mirror group
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_ReturnMirrorInfo(
   LONG                          mirrorObjectID,
   struct MM_MirrorInfoStruct    *info);
```

Parameters

mirrorGroupObjectID

(IN) The Media Manager object ID of the mirror object for which mirror information is requested. This object does not have to be reserved by the application prior to this call.

info

(IN) The pointer to the MM_MirrorInfoStruct structure.

Return Values

MM_OK

Function completed successfully.

MM_INVALID_OBJECT

mirrorObjectID was invalid (not in the database or not a Mirror Object) or it did not have either a Hot Fix Object as a parent or a Partition Object for a grandparent.

MM_PARAMETER ERROR

One or more parameters passed is not what was expected.

MM_FAILURE

The function failed to return the mirror information requested.

MM_ReturnPartitionTableInfo

Returns partition information for a selected Device Object
Classification: 3.12, 4,x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <partapi.h>

LONG MM_ReturnPartitionTableInfo(
   LONG                                deviceObjectHandle,
   LONG                               *partitionTableStatus,
   LONG                               *alignment,
   LONG                               *areaStart,
   LONG                               *areaLength,
   LONG                               *partitionCount,
   struct MM_PartitionInfoStruct      *partitionInfo);
```

Parameters

deviceObjectHandle

(IN) The Media Manager object handle of a reserved Device Object.

partitionTableStatus

(IN) See Media Manager: Structures for examples of this status.

alignment

(IN) Number of sectors per cylinder (i.e., number of Heads * number of Sectors per Track).

areaStart

(IN) The beginning sector number where the first partition will be placed.

areaLength

(IN) The number of useable sectors on the media.

partitionCount

(IN) Number of partition elements on the media.

partitionInfo

(IN) See MM_PartitionInfoStruct for structure definition. Memory must be allocated for 8 structure elements. **Caution:** Memory will be overwritten if this is not accounted for properly.

Return Values

`MM_OK`

Function completed successfully.

`MM_INVALID_OBJECT`

deviceObjectHandle was invalid.

`MM_INVALID_MODE`

The object was not active.

`MM_RESERVATION CONFLICT`

The Device Object was not reserved by the application prior to the request

`MM_PARAMETER ERROR`

One or more parameters passed is not what was expected.

`MM_FAILURE`

The function failed to return the partition table information requested.

`MM_IO_ERROR`

The I/O request to the selected device failed.

MM_Set_Object_Attribute

Sets an object's attributes
Classification: 3.12, 4,*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Set_Object_Attribute(
   LONG   objectHandle,
   LONG   attributeID,
   LONG   length,
   void   *info);
```

Parameters

objectHandle

This is the ID of the object for which an attribute will be set.

attributeID

This is the ID of the attribute to be set. The attribute ID is the first four bytes (swapped) of the attribute name. See Attribute IDs for a list.

length

This is the size of the buffer containing object attribute information.

info

This points to a structure that contains object specific information (such as the object's attributes). The format of this structure depends upon the object class and application type (see Media Manager: Structures).

Return Values

`MM_OK`

Function completed successfully.

`MM_MEMORY_ALLOCATION_ERROR`

Memory allocation failed.

`MM_PARAMETER_ERROR`

objectHandle was not found in the database or *attributeID*, *length*, and/or *info* were not set to useable values.

Remarks

This function sets an object's attributes to a desired value. The attributes that may be set for an object depend on the object's type.

MM_Set_Unload_Semaphore

Postpones the deactivation of a device after receiving a driver deactivation alert

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG  MM_Set_Unload_Semaphore(
   LONG   *currentInstance);
```

Parameters

currentInstance

Returned by the OS to uniquely identify the current semaphore (in cases where there are nested semaphores)

Return Values

0

Function completed successfully.

-1

Out of context

Remarks

Two of the application alerts defined in Function/Completion Code Definitions are used to indicate unload/deactivate conditions. Either the OS issued a driver unload request (GOING_TO_BE_DEACTIVATED), or an application issued a deactivate command (OBJECT_BEING_DEACTIVATED).

The **MM_Set_Unload_Semaphore** function allows the application to postpone the deactivation of the device after receiving one of these alerts until the application has been able to flush tables and buffers.

Example sequence of events:

1. The OS alerts the application with one of the above alert messages (0x0001, or 0x0002).
2. The application calls MM_Set_Unload_Semaphore.

3. The application spawns a process (or wakes up an existing process) to flush buffers.

4. The spawned process calls MM_Release_Unload_Semaphore after completing clean up.

This sequence of events coordinates the unloading of the driver with applications that depend on the devices that are to be deactivated.

The application cannot call this function from anywhere outside the context of the alert, but it can call this function several times from the same alert function. Be sure to call the "remove" function as many times as the "set" function is called.

MM_Special_Object_Blocking_IO

This is a blocking function. It issues I/O requests to the object specified by objectID and returns when the action code is complete. This call is only valid during media identification and labeling.

Classification: 3.12, 4,x

SMP Aware: No

Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Special_Object_Blocking_IO(
   LONG   *returnParameter,
   LONG   objectID,
   LONG   actionCode,
   LONG   parameter1,
   LONG   parameter2,
   LONG   parameter3,
   LONG   bufferLength,
   void   *buffer);
```

Parameters

returnParameter

This parameter points to a 4 byte location for the information.

objectID

This parameter is the object ID of a device with mounted media.

actionCode

This parameter indicates which action code is to be performed. (See Chapter 3 and Chapter 4).

parameter 1,2,3

The meaning of these parameters depends upon the action code specified. (See Chapter 3 and Chapter 4).

bufferLength

This parameter specifies the length of the buffer.

buffer

This parameter points to data or control buffers passed to the device. The buffer size is specified by *bufferLength*.

Return Values

`MESSAGE_PROCESSED`

Function completed successfully.

`MESSAGE_POSTPONE`

The operation failed because of memory allocation errors; you should retry the operation later.

`MESSAGE_INVALID_PARAMETERS`

osReserveHandle was not found in the database.

`MESSAGE_ABORTED_CLEAN`

Remarks

The **MM_Set_Unload_Semaphore** function allows the application to postpone the deactivation of the device after receiving one of these alerts until the application has been able to flush tables and buffers.

MM_Unregister_Application

Deletes an application registration from the Media Manager
Classification: 3.12, 4.*x*
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG  MM_Unregister_Application (
   LONG   osApplicationHandle,
   LONG   applicationHandle);
```

Parameters

osApplicationHandle

This parameter is the same OS application handle passed to **MM_Register_Application** on a successful registration.

applicationHandle

This parameter is the application handle that was passed to **MM_Register_Application**.

Return Values

`MM_OK`

Function completed successfully.

`MM_PARAMETER_ERROR`

osApplicationHandle and/or *applicationHandle* were invalid.

MM_Unregister_Identification_Routines

Unregisters identification functions from the Media Manager
Classification: 3.12, 4.x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Unregister_Identification_Routines(
   LONG   osidentifyHandle,
   LONG   identifyHandle);
```

Parameters

osidentifyHandle

The *osIdentifyHandle* returned by MM_Register_Identification_Routines.

identifyHandle

The Novell assigned media format type used to distinguish between identification functions.

Return Values

`MM_OK`

Function completed successfully.

`MM_PARAMETER_ERROR`

osApplicationHandle and/or *applicationHandle* were invalid.

MM_Unregister_Notify_Routine

Unregisters a notification function from the Media Manager
Classification: 3.12, 4.x
SMP Aware: No
Service: Media Manager

Syntax

```
#include <nwmediam.h>

LONG MM_Unregister_Notify_Routine(
   LONG   osNotifyHandle,
   LONG   notifyHandle);
```

Parameters

osNotifyHandle

The *osNotifyHandle* returned by *MM_Register_Notify_Routine*.

notifyHandle

The *notifyHandle* passed to **MM_Register_Notify_Routine**.

Return Values

`MM_OK`

Function completed successfully.

`MM_PARAMETER_ERROR`

osNotifyHandle and/or *notifyHandle* had invalid values.

File Service Group

Media Manager: Structures

AdapterInfoDef

Contains information about an adapter object
Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct  AdapterInfoDef
{
   BYTE     systemType;
   BYTE     processorNumber;
   WORD     uniqueTag;
   LONG     systemNumber;
   LONG     devices[32];
   struct IOConfigurationStructure   configInfo;
   BYTE     driverName[36];
   BYTE     systemName[64];
   LONG     numberOfDevices;
   LONG     reserved[7];
};
```

Fields

systemType

Contains the driver type assigned by Novell.

processorNumber

Contains the server number for SFT III.

uniqueTag

devices

Contains the object IDs of dependent devices or changers.

configInfo

Contains I/O port information such as share flags, DMA address, and port address.

driverName

Contains the name of the NLM as a length-preceded, NULL-terminated string.

systemName

Contains the system name as a length-preceded ASCII string.

numberOfDevices

Contains the number of devices attached to this adapter.

reserved

Reserved by Novell.

Remarks

Returned by **MM_Return_Object_Specific_Info**.

AttributeInfoDef

Contains information about an attribute of an object
Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct AttributeInfoDef
{
   BYTE   name[64];
   WORD   attributeType;
   WORD   settableFlag;
   LONG   nextAttributeID;
   LONG   attributeSize;
};
```

Fields

name

Contains the name of the attribute type as a length-preceded, NULL-terminated string.

attributeType

Contains the attribute type (see Attribute Types).

settableFlag

Indicates whether the attribute can be set with **MM_Set_Object_Attribute**:

0 = Cannot be set using **MM_Set_Object_Attribute**
1 = Can be set using **MM_Set_Object_Attribute**

nextAttributeID

Contains the ID of the next available object attribute.

attributeSize

Contains the size of *attributeType*.

Remarks

Used by **MM_Return_Objects_Attributes**.

ChangerInfoDef

Contains information about a changer
Service: Media Manager
Defined In: nwmediam.h

Structure

Not at Run-Time

```
struct  ChangerInfoDef
{
   LONG   numberOfDevices;
   LONG   numberOfSlots;
   LONG   numberOfMailSlots;
   LONG   reserved[8];
   LONG   slotMappingTable[1];
};
```

At Run-Time

```
struct  ChangerInfoDef
{
   LONG   numberOfDevices;
   LONG   numberOfSlots;
   LONG   numberOfMailSlots;
   LONG   reserved[8];
   LONG   deviceObjectID[numberOfDevices];
   LONG   slotObjectID[numberOfSlots];
   LONG   mailSlotsObjectID[numberOfMailslots];

};
```

Fields

numberOfDevices

Contains the number of devices in the changer.

numberOfSlots

Contains the total number of slots available for media.

numberOfMailSlots

Contains the number of slots where operators insert and remove media.

reserved

Reserved by Novell.

slotMappingTable

deviceObjectID

Contains arrays of object.

slotObjectID

Contains IDs of each type.

mailSlotsObjectID

Contains IDs of objects in the changer.

Remarks

This structure is prototyped in MM.H as shown in the first structure definition below because the *numberOfDevices*, *numberOfSlots*, and *numberOfMailslots* parameters are not known before run time. At run-time, the second definition of the structure will apply.This structure is returned by **MM_Return_Object_Specific_Info** when the application requests information about an autochanger object. The parameters *numberOfDevices*, *numberOfSlots*, and *numberOfMailSlots* are mutually exclusive of each other.

DeviceInfoDef

Contains information about device objects.
Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct  DeviceInfoDef
{
   LONG   status;
   BYTE   controllerNumber;
   BYTE   driveNumber;
   BYTE   cardNumber;
   BYTE   systemType;
   BYTE   accessFlags;
   BYTE   type;
   BYTE   blockSize;
   BYTE   sectorSize;
   BYTE   heads;
   BYTE   sectors;
   WORD   cylinders;
   LONG   capacity;
   LONG   mmAdapterNumber;
   LONG   mmMediaNumber;
   BYTE   rawName[40];
   LONG   reserved[8];
};
```

Fields

status

Contains Media Manager object status. The bits represent activated, loaded, and so on.

controllerNumber

Contains the ID of the adapter board.

driveNumber

Contains the device number assigned by the driver.

cardNumber

Contains the card number assigned by the driver.

systemType

Contains the driver type.

accessFlags

Indicates such access information as removable, read-only, write

sequential, dual port, HotFixInhibit or MirrorInhibit.

type

blockSize

Contains the size of group of sectors to be transferred at once (in bytes).

sectorSize

Contains the requested size for sectors (in bytes). The default is 512 bytes.

heads

Parameter 1 for device objects.

sectors

Parameter 2 for device objects.

cylinders

Parameter 3 for device objects.

capacity

Contains the total capacity of the device in sectors.

mmAdapterNumber

Contains the Media Manager object ID for the adapter board.

mmMediaNumber

Contains the Media Manager object ID for the media in the device.

rawName

Contains the device name passed from the driver.

reserved

Reserved by Novell.

Remarks

Returned by **MM_Return_Object_Specific_Info**.

GenericInfoDef

Contains information about a fixed device object
Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct GenericInfoDef
{
   struct MediaInfoDef   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;
}
```

Fields

mediaInfo

Identifies media.

mediaType

Identifies the media type (CD-ROM, changer disk, and so on).

cartridgeType

Contains the cartridge or magazine type that the device can use.

unitSize

Contains the number of bytes per sector.

blockSize

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

capacity

Contains the maximum number of sectors on the device.

preferredUnitSize

Contains the data unit size preferred by the device (up to 1 K for formatted devices).

name

Contains the name as a length-preceded ASCII string.

type

Contains the database object type (that is, mirror, partition, magazine, and so on).

functionMask

Contains a bitmap of the functions supported by the device (20h - 2Fh).

controlMask

Contains the Media Manager function (0 - 1F).

parentCount

Contains the number of objects that the device depends on (usually one).

siblingCount

Contains the number of objects with common dependencies.

childCount

Contains the number of objects that depend on the device.

specificInfoSize

Contains the size of data structures to be returned.

objectUniqueID

Contains the object ID for this instance ofGenericInfoDef.

mediaSlot

Identifies which slot the media occupies.

Remarks

Returned by **MM_Return_Object_Generic_Info**.

HotFixInfoDef

Contains information about a Hot Fix object
Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct  HotFixInfoDef
{
   LONG   hotFixOffset;
   LONG   hotFixIdentifier;
   LONG   numberOfTotalBlocks;
   LONG   numberOfUsedBlocks;
   LONG   numberOfAvailableBlocks;
   LONG   numberOfSystemBlocks;
   LONG   reserved[8];
};
```

Fields

hotFixOffset

Contains the offset of data (Hot Fix begins at 0000h).

hotFixIdentifier

Contains the unique identifier created when the partition undergoes Hot Fix.

numberOfTotalBlocks

Contains the total number of 4 K blocks available in the Hot Fix area.

numberOfUsedBlocks

Contains the number of 4 K blocks that contain redirected data.

numberOfAvailableBlocks

Contains the number of blocks in the Hot Fix area that are not allocated.

numberOfSystemBlocks

Contains the number of blocks used for internal Hot Fix tables and bad blocks.

reserved

Reserved by Novell.

Remarks

Returned by **MM_Return_Object_Specific_Info**.

InsertRequestDef

Used for handling requests from an application driver for a particular piece of media within a changer

Service: Media Manager

Defined In: nwmediam.h

Structure

```
struct InsertRequestDef
{
   LONG   deviceNumber;
   LONG   mailSlot;
   LONG   mediaNumber;
   LONG   mediaCount;
};
```

Fields

deviceNumber

Contains the number of the device in the media changer that media move in or out of.

mailSlot

Identifies the slot in the media changer where operators insert and remove media.

mediaNumber

Contains the slot number.

mediaCount

Contains the number of media in the media changer.

MagazineInfoDef

Contains information about a magazine object
Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct  MagazineInfoDef
{
   LONG   numberOfSlots;
   LONG   reserved[8];
   LONG   slotMappingTable[numberOfSlots];
};
```

Fields

numberOfSlots

Contains the number of slots in the magazine plus one (for the device).

reserved

Reserved by Novell.

slotMappingTable

Contains a byte table of all slots. The following indicates status:

0 = empty
non-zero = has media

slotMappingTable[0] is the location that indicates the media status for the device, and *slotMappingTable*[1] through *slotMappingTable*[*numberOfSlots*] represent all the slots of the magazine.

Remarks

Returned by **MM_Return_Object_Specific_Info**.

MappingInfo

Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct  MappingInfo
{
   LONG   parentCount;
   LONG   siblingCount;
   LONG   childCount;
   LONG   parentObjectID[parentCount];
   LONG   siblingObjectIDs[siblingCount];
   LONG   childObjectIDs[childCount];
};
```

Fields

parentCount

Contains the number of objects that the object depends on.

siblingCount

Contains the number of objects that depend on the parent of an object.

childCount

Contains the number of objects that depend on this device.

parentObjectID

Contains an array of parent object IDs.

siblingObjectIDs

Contains an array of sibling object IDs.

childObjectIDs

Contains an array of child object IDs.

Remarks

This structure is not prototyped in MM.H because the*parentCount*, *siblingCount*, and *childCount* parameters are not known before run time. *MappingInfo* is used to hold the information returned by **MM_Return_Object_Mapping_Info**. The minimum possible size of this structure is the first three LONGs shown, which would occur if there are no parents, siblings, or children. For any existing object,*siblingCount* will always be at least 1, since each object is its own sibling.

NOTE: If the device is a magazine, this structure will list one child. That child will be the magazine object. To obtain the list of media

associated with this magazine, call
MM_Return_Object_Mapping_Info for this magazine object.

MediaInfoDef

Identifies a physical media item
Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct  MediaInfoDef
{
   BYTE   label[64];
   LONG   identificationType;
   LONG   identificationTimeStamp;
};
```

Fields

label

Contains an ASCII string that identifies the media.

identificationType

Contains the number assigned to the media by Novell.

identificationTimeStamp

Contains the UNIX timestamp.

Remarks

This structure is used to identify or create a physical media item in **MM_Create_Media_Object**. *MediaInfoDef* is also passed when labeling new media. It is filled in when registering ID functions.

MediaRequestDef

Used for handling requests from an application driver for a particular piece of media within a changer

Service: Media Manager

Defined In: nwmediam.h

Structure

```
struct  MediaRequestDef
{
   LONG   deviceNumber;
   LONG   mailSlot;
   LONG   mediaNumber;
   LONG   mediaCount;
};
```

Fields

deviceNumber

Contains the object ID of the device within the media changer that media moves in and out of.

mailSlot

Contains the slot number (slot ID) of the slot in the media changer where operators insert and remove media.

mediaNumber

Contains the slot number (object ID).

mediaCount

Contains the total number of media in the media changer.

MirrorInfoDef

Contains information about a mirror object
Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct  MirrorInfoDef
{
   LONG   mirrorCount;
   LONG   mirrorIdentifier;
   LONG   mirrorMembers[8];
   BYTE   mirrorSyncFlags[8];
   LONG   reserved[8];
};
```

Fields

mirrorCount

Contains the number of partitions in the mirror group.

mirrorIdentifier

Contains the unique number assigned when the mirror group was created.

mirrorMembers

Contains the object IDs of Hot Fix objects in the mirror group.

mirrorSyncFlags

Indicates partitions that have current data:

0 = old data
nonzero = current data

reserved

Reserved by Novell.

Remarks

Returned by **MM_Return_Object_Specific_Info**.

MM_MirrorInfoStruct

Contains information about a mirror group
Service: Media Manager
Defined In: nwpartapi.h

Structure

```
struct  MM_MirrorInfoStruct
{
   WORD   status;
   WORD   mirrorcount;
   LONG   mirroridentifier;
   LONG   mirrormembers[16];
   LONG   mirrorpercentage;
   LONG   reserved[7];
};
```

Fields

status

Contains an OR of bits listed in the "Remarks" section below.

mirrorcount

Specifies the number of partitions including this one in the mirror group.

mirroridentifier

Specifies the partition number of the original partition in the mirror group.

mirrormembers

Contains an array of the partitions that make up this mirror group.

remirrorpercentage

Specifies the percentage of the partition that is mirrored (range = 0 - 99)

reserved

Reserved by Novell.

Remarks

The status field is an OR of the following bits:

Constant	Hex Value	Meaning

MIRRORGROUPINSYNCH	0x01	The mirror group is in sync.
MIRRORGROUPALLHERE	0x02	All parts of the mirror are present.
MIRRORGROUPOPERATIONAL	0x04	The mirror group is operational.
IAMINSYNCH	0x10	Any partition in the group is in sync.
MIRRORGROUPBEINGREMIRRORED	0x40	The mirror group is being remirrored.

MM_PartitionInfoStruct

Contains information about a media partition
Service: Media Manager
Defined In: nwpartapi.h

Structure

```
struct  MM_PartitionInfoStruct
{
   BYTE   partitionName[32];
   LONG   beginningSector;
   LONG   numberOfSectors;
   LONG   objectID;
   LONG   bootFlag;
};
```

Fields

partitionName

Specifies the type of partition as an ASCII string (for example "NetWare Partition").

beginningSector

Specifies the beginning sector number of the physical partition.

numberOfSectors

Specifies the number of sectors in the physical partition.

objectID

Specifies the Media Manager ObjectID of a defined partition for this physical partition, if any (-1 means none).

bootFlag

Specifies the boot flag of the partition entry.

Remarks

Used in **MM_ReturnPartitionTableInfo**.

PartitionInfoDef

Contains information about a partition object
Service: Media Manager
Defined In: nwmediam.h

Structure

```
struct  PartitionInfoDef
{
   LONG   partitionerType;
   LONG   partitionType;
   LONG   partitionOffset;
   LONG   partitionSize;
   LONG   reserved[8];
};
```

Fields

partitionerType

Indicates the partition scheme (that is, DOS, IBM, and so on).

partitionType

Indicates the partition type. Only the lower byte is important.

partitionOffset

Contains the beginning sector number of the partition.

partitionSize

Contains the number of sectors in the partition. The default is 512 KB.

reserved

Reserved by Novell.

Remarks

Returned by **MM_Return_Object_Specific_Info**.

Name Space

Name Space: Guides

Name Space: Task Guide

Accessing Huge Name Space Information

Accessing Name Space Data Streams

Additional Links

Name Space: Functions

Name Space: Structures

Parent Topic:

Name Space: Guides

Name Space: Concept Guide

Name Space Introduction

Name Space Naming Conventions

Primary Name Space Entry Information

Name Space Specific Information

Name Space Entry Bit Masks

Name Space Bit Mask

DOS Name Space Bit Mask

General Name Space Functions

General Name Space Functions

Primary Name Space Entry Information Functions

Name Space Specific Information Functions

Additional Links

Name Space: Functions

Name Space: Structures

Parent Topic:

Name Space: Guides

Name Space Specific Information

Name space specific information is maintained by the NLM that implements the name space. Much of this information may not be accessible as primary information. For example, huge data information is name space specific and must be returned by special requests

Consequently, Name Space includes specialized functions for accessing name space specific information. This approach requires a detailed understanding of the particular name space and the entry information it maintains.

Name space specific information is accessed by calling **NWReadNSInfo** and **NWWriteNSInfo**. Both functions refer to the entry using a NetWare® entry index, which is maintained as NW_IDX. To initialize NW_IDX, call **NWGetDirectoryBase** and pass both a DOS directory entry (handle/path) and the target name space.

Name Space Entry Bit Masks

Name Space Bit Mask

DOS Name Space Bit Mask

Accessing Huge Name Space Information

Parent Topic:

Name Space: Guides

Name Space: Tasks

Accessing Huge Name Space Information

The huge information bit mask indicates large data items (between 256 and 65,535 bytes) associated with a name space entry. Call **NWReadExtendedNSInfo** and **NWWriteExtendedNSInfo** to access huge information. An operation on huge data must include the huge information bit mask for the name space, the length of the huge data, and a huge state information variable. This last value is maintained by the server and is used to coordinate the transmission of huge data.

Parent Topic:

Name Space Specific Information

Accessing Name Space Data Streams

Call **NWOpenNSEntry** to open or create a data stream for a name space entry. Opening a data stream returns a file handle you can use for read/write operations. Currently, NetWare® defines three data streams:

0 DOS
1 MAC
2 FTAM

You can open an existing data stream for reading and writing through the DOS name space, or you can create a new data stream within a specified name space. When an entry has more than one data stream, as is the case for Macintosh files, specify which stream to open.

When creating a data stream, specify the open/create mode. Possible modes include creating the file, replacing the file, opening the file, or doing nothing. (A directory can only be opened.) You can also specify the entry's attributes and the access rights under which the entry is opened. File entries and directory entries have their own respective attributes; specify them accordingly.

The following access rights can be defined for an open data stream:

0001h = AR_READ
0002h = AR_WRITE
0004h = AR_DENY_READ
0008h = AR_DENY_WRITE

0010h = AR_COMPATIBILITY
0040h = AR_WRITE_THROUGH
0100h = AR_OPEN_COMPRESSED

AR_READ gives you exclusive access to the data stream.

The AR_OPEN_COMPRESSED structure is available only on NetWare servers running NetWare 4.0™ and above. It allows a data stream to be opened in a compressed state. (Normally, the server decompresses a compressed data stream as it transmits the data to a client.)

Parent Topic:

Name Space: Guides

Name Space: Concepts

Attribute Values

The following are attribute values:

C Value	Pascal Value	Value Name
0x00000000 L	$00000000	A_NORMAL
0x00000001 L	$00000001	A_READ_ONLY
0x00000002 L	$00000002	A_HIDDEN
0x00000004 L	$00000004	A_SYSTEM
0x00000008 L	$00000008	A_EXECUTE_ONLY
0x00000010 L	$00000010	A_DIRECTORY
0x00000020 L	$00000020	A_NEEDS_ARCHIVED
0x00000080 L	$00000080	A_SHAREABLE
0x00001000 L	$00001000	A_TRANSACTIONAL
0x00002000 L	$00002000	A_INDEXED
0x00004000 L	$00004000	A_READ_AUDIT
0x00008000 L	$00008000	A_WRITE_AUDIT
0x00010000 L	$00010000	A_IMMEDIATE_PURGE
0x00020000 L	$00020000	A_RENAME_INHIBIT
0x00040000 L	$00040000	A_DELETE_INHIBIT

0x00080000 L	$00080000	A_COPY_INHIBIT
0x00400000 L	$00400000	A_FILE_MIGRATED
0x00800000 L	$00800000	A_DONT_MIGRATE
0x02000000 L	$02000000	A_IMMEDIATE_COMPRESS
0x04000000 L	$04000000	A_FILE_COMPRESSED
0x08000000 L	$08000000	A_DONT_COMPRESS
0x20000000 L	$20000000	A_CANT_COMPRESS

DOS Name Space Bit Mask

The interpretation of the name space bit mask depends on which name space you are querying. For example, the DOS name space defines the following bits:

Bit	Definition	Type	Order
0	Modify Name[13]	nuint 8	
1	File Attributes	nuint 32	Lo-Hi
2	Create Date	nuint 16	Lo-Hi
3	Create Time	nuint 16	Lo-Hi
4	Owner ID	nuint 32	Hi-Lo
5	Archive Date	nuint 16	Lo-Hi
6	Archive Time	nuint 16	Lo-Hi
7	Archive ID	nuint 32	Hi-Lo
8	Modify Date	nuint 16	Lo-Hi

9	Modify Time	nuint 16	Lo-Hi
10	Modify ID	nuint 32	Hi-Lo
11	Last Accessed Date	nuint 16	Lo-Hi
12	Inheritance Rights	nuint 32	Lo-Hi
13	Maximum Space	nuint 32	Lo-Hi
14-31	Reserved		

Under DOS, bit 0 represents the modify name. This is generally the case in other name spaces also. The modify name is read-only; don't attempt to modify it.

Parent Topic:

Name Space Specific Information

General Name Space Functions

These functions return general information concerning name spaces.

Function	Comment
NWGetNSLoadedList	Returns a list of numerals identifying the name spaces loaded on a particular volume.
NWGetOwningNameSpace	Returns the name space that created the specified directory entry.
NWGetNSPath	Returns the full path for an entry in a specified name space. (For name spaces that use long names, a complete entry path could potentially require a very large amount of space.)
NWNSGetDefaultNS	Returns the default name space.

Parent Topic:

Name Space: Guides

Name Space Entry Bit Masks

NetWare® uses a generic mechanism to represent the format of name space specific entry information. Query the NetWare server by calling **NWGetNSInfo** to find the format for a particular name space. **NWGetNSInfo** returns a set of bit masks as NW_NS_INFO. The structure indicates the size and arrangement of name space specific information.

Parent Topic:

Name Space Specific Information

Name Space Bit Mask

NSInfoBitMask in NW_NS_INFO indicates all valid data items for an entry in the name space. **NWGetNSInfo** initializes the bit masks for a specific name space and computes the value of *NSInfoBitMask*.

NSInfoBitMask is derived by combining the fixed and reserved masks through a logical OR operation.

After NW_NS_INFO is initialized, use it in subsequent calls to **NWReadNSInfo** and **NWWriteNSInfo** to read or modify name space specific entry information.

Parent Topic:

Name Space Specific Information

Name Space Introduction

NetWare® name spaces are implemented as NetWare Loadable Module™ (NLM™) applications. Name Spaces allow NetWare 3.11 and above servers to store files in formats compatible with a workstation's local file system. For example, installing the Macintosh* name space allows Macintosh workstations to use Macintosh file conventions when working with network files. Although NetWare's primary name space is DOS, NetWare also supports name spaces for OS/2, Macintosh, NFS*, and FTAM files.

After the NLM is loaded on a server, support for the name space must be enabled on a volume-by-volume basis. Name space entry information can include the entry's name, its attributes, significant dates and times, the owner ID, and so on.

Name Space provides a generic interface to name space entries and associated data streams. However, Name Space isn't the only method for accessing entries in a particular name space. DOS and OS/2 entries can be accessed through File, and Macintosh files can be accessed through AFP.

Name Space provides access to three types of data: primary data, which is the data that is available no matter which name space you are using;

specific data, which is the data specific to the name space you are using, and the actual file data.

Parent Topic:

Name Space: Guides

Name Space Naming Conventions

NetWare® currently supports five name spaces. They are identified by numeric values.

0 DOS
1 MAC
2 NFS
3 FTAM
4 OS/2
4 Long

Each name space has its own conventions for naming entries:

DOS names can have up to eight upper-case characters followed by a period and up to three more upper-case characters.

Macintosh names can be up to 32 characters long including all upper and lower case printable characters, with the exception of the colon (:).

NFS names can be up to 256 mixed-case characters long.

FTAM names can be up to 256 lower-case characters long.

OS/2 and Long names can be up to 255 mixed-case characters long. Name Space Services make OS/2 and Long names look like and follow the same rules as HPFS names.

DOS names remain the same in an OS/2 environment. NetWare uses a shortening algorithm to convert OS/2 names for use in a DOS environment. To avoid ambiguous names, this algorithm may designate a DOS filename that doesn't match the first eight characters of the OS/2 name.

Parent Topic:

Name Space: Guides

Name Space Specific Information Functions

These functions deal with name-space specific information.

Function	Comment

NWGetDirectoryBase	Obtains a directory base for a name space entry.
NWGetNSInfo	Returns the information format for a name space.
NWNSGetMiscInfo	Obtains miscellaneous information for a name space entry.
NWReadExtendedNSInfo	Reads huge information for an entry.
NWReadNSInfo	Reads name space-specific information for an entry.
NWWriteExtendedNSInfo	Modifies huge information for an entry.
NWWriteNSInfo	Modifies name space-specific information for an entry.

Parent Topic:

Name Space: Guides

Primary Name Space Entry Information

As the primary NetWare® name space, the DOS name space performs a special role in the NetWare file system. All entries are represented in the DOS name space no matter what name space actually "owns" them. Consequently, if you create an entry in a name space other than DOS, you can still access the primary entry information from the DOS name space.

This primary NetWare information is extended beyond DOS to accommodate Macintosh data, including information such as the number of data streams (forks) and extended attributes (Finder information).

In addition to letting you read an entry's primary information in the DOS name space, Name Space Services enable you to read and modify this information in the name space that the entry was created in. The primary information in the owning name space varies little from what appears in the DOS name space. However, it does include the file's long name, which isn't available in the DOS name space.

Primary name space information includes the following items:

Entry name

Entry attributes

Space allocation

Data stream sizes

Dates and time of events

Inherited rights mask

Extended attribute data

Reference ID

Volume Number

NW_ENTRY_INFO contains primary name space information. The structure is filled in by **NWGetNSInfo** or **NWScanNSEntryInfo**. Requests for primary name space information are accompanied by a return information mask, which allows you to specify which portions of NW_ENTRY_INFO you want filled in. The following table shows which fields in NW_ENTRY_INFO are affected by bit flags in the return information mask.

Table auto. Return Information Mask Values

Value	Flag	Fields
0x0001L	IM_ENTRY_NAME	nameLength entryName
0x0002L	IM_SPACE_ALLOCATED	spaceAlloc
0x0004L	IM_ATTRIBUTES	attributes flags
0x0008L	IM_SIZE	dataStreamSize
0x0010L	IM_TOTAL_SIZE	totalStreamSize
0x0020L	IM_EA	EADataSize EAKeyCount EAKeySize
0x0040L	IM_ARCHIVE	archiveTime archiveDate archiveID
0x0080L	IM_MODIFY	modifyTime modifyDate modifierID lastAccessDate
0x0100L	IM_CREATION	creationTime creationDate creatorID
0x0200L	IM_OWNING_NAMESPACE	NSCreator
0x0400L	IM_DIRECTORY	dirEntNum DosDirNum

		volNumber
0x0800L	IM_RIGHTS	inheritedRightsMask

Parent Topic:

Name Space: Guides

Primary Name Space Entry Information Functions

These functions deal with primary entry information for a name space.

Function	Comment
NWAllocTempNSDirHandle2	Allocates a directory handle in a name space for the specified entry. The new directory handle doesn't need to be in the same name space as the original entry.
NWDeleteNSEntry	Erases the specified files from the file server.
NWGetLongName	Reads an entry's name in the specified name space.
NWGetNSEntryInfo	Returns primary information for a name space entry.
NWNSRename	Renames a name space entry. Under NetWare® 4.x, this function can rename an entry in a specific name space without affecting the name in other name spaces.
NWOpenCreateNSEntry	Creates a name space entry.
NWOpenDataStream	Opens or creates a data stream and returns a file handle to it.
NWOpenNSEntry	Opens a name space entry.
NWScanNSEntryInfo	Performs a file scan operation returning primary information for files matching the search mask.
NWSetLongName	Renames a name space entry.
NWSetNSEntryDOSInfo	Modifies the DOS information associated with an entry.

Parent Topic:

Name Space: Guides

Name Space: Functions

GetDataStreamName

Returns information about data streams
Local Servers: nonblocking
Remote Servers: N/A
Classification: 3.x, 4.x
SMP Aware: No
Service: Name Space

Syntax

```
#include <nwfile.h>

int GetDataStreamName (
   int    volume,
   BYTE   dataStream,
   char   *dataStreamName,
   int    *numberOfDataStreams);
```

Parameters

volume

(IN) Specifies the number of the volume for which the data stream name is desired.

dataStream

(IN) Specifies the number of the data stream whose name is desired.

dataStreamName

(OUT) Receives the ASCII name of the data stream.

numberOfDataStreams

(OUT) Receives the number of data streams supported by the server.

Return Values

This function returns TRUE if the name space that defines the specified data stream is loaded on the volume. It returns FALSE if support is not loaded. If the data stream does not exist, this function returns a value of -1.

Remarks

The name of the specified data stream is returned, as well as the total number of data streams available. The function return also indicates whether the specified data stream has support on the volume.

The *dataStream* parameter is a data stream number. The defined data streams follow:

0	Primary Data Stream (corresponds to DOS)
1	Macintosh Resource Fork
2	FTAM Extra Data Fork

GetNameSpaceName

Returns the name of a specified name space and the number of name spaces currently supported by NetWare

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x

SMP Aware: Yes

Service: Name Space

Syntax

```
#include <nwfile.h>

int GetNameSpaceName (
   int    volume,
   LONG   nameSpace,
   char   *name,
   int    *numberOfNameSpace);
```

Parameters

volume

(IN) Specifies the volume for which name space information is desired.

nameSpace

(IN) Specifies the number of the name space whose name is desired.

name

(OUT) Receives the name of the name space in ASCIIZ string (buffer length should be 32 bytes).

numberOfNameSpace

(OUT) Receives the number of name spaces currently supported by NetWare.

Return Values

-1	Specified name space does not exist.
0	Name space driver is not loaded.
1	Name space driver is loaded but is not supported on the specified volume.
2	Name space driver is loaded and supported on the specified volume.

Remarks

The five name spaces that are currently available are:

0	DOS
1	MACINTOSH
2	NFS
3	FTAM
4	OS2
5	NT

See Also

FEGetOriginatingNameSpace, **SetCurrentNameSpace**, **SetTargetNameSpace**

NWAllocTempNSDirHandle (obsolete 12/96)

Assigns a temporary directory handle for *dirHandle/path* in the same name space as *dirHandle/path* but is now obsolete. Call NWAllocTempNSDirHandle2 instead.

NetWare Server: 3.11, 3.12, 4.x

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

Service: Name Space

Syntax

```
#include <nwnamspc.h>
or
#include <nwcalls.h>);

NWCCODE N_API NWAllocTempNSDirHandle (
   NWCONN_HANDLE   conn,
   nuint8          dirHandle,
   pnstr8          path,
   nuint8          nameSpc,
   nuint8 N_FAR   *newDirHandle);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWAllocTempNSDirHandle
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   path : pnstr8;
   nameSpc : nuint8;
   Var newDirHandle : nuint8
) : NWCCODE;
```

Parameters

conn

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

dirHandle

(IN) Specifies the directory handle associated with the desired directory path.

path

(IN) Points to an absolute path, (or relative if *dirHandle* is non-zero), with which *newDirHandle* is to be associated.

nameSpc

(IN) Specifies the name space of *dirHandle/path*.

newDirHandle

(OUT) Points to the new directory handle.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89BF	INVALID_NAME_SPACE

Remarks

To allocate *dirHandle* in a different name space, call **NWAllocTempNSDirHandle2**. In order for **NWAllocTempNSDirHandle (obsolete 12/96)** to perform properly it is required that the specified name space be loaded on the server being accessed. For example, if an OS/2 workstation calls **NWAllocTempNSDirHandle (obsolete 12/96)** with the *nameSpace* parameter set to OS/2, the *newNameSpace* parameter set to DOS, and the server being accessed does not have the OS/2 name space loaded, **NWAllocTempNSDirHandle (obsolete 12/96)** will not work.

To allocate OS/2, *dirHandle/path* must use the OS/2 format.

The directory handles allocated by **NWAllocTempNSDirHandle (obsolete 12/96)** are automatically deallocated when the task terminates or by calling **NWDeallocateDirectoryHandle**.

NCP Calls

0x2222 87 12 Allocate Short Directory Handle

See Also

NWAllocTempNSDirHandle2, **NWDeallocateDirectoryHandle**

NWAllocTempNSDirHandle2

Assigns a temporary directory handle in the specified name space
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWAllocTempNSDirHandle2 (
   NWCONN_HANDLE   conn,
   nuint8          dirHandle,
   pnstr8          path,
   nuint8          nameSpc,
   pnuint8         newDirHandle,
   nuint8          newNameSpace);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWAllocTempNSDirHandle2
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   path : pnstr8;
   namSpc : nuint8;
   newDirHandle : pnuint8;
   newNameSpace : nuint8
) : NWCCODE;
```

Parameters

conn

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

dirHandle

(IN) Specifies the directory handle associated with the desired directory path.

path

(IN) Points to an absolute path, (or relative if *dirHandle* is non-zero),

with which *dirHandle* is to be associated.

namSpc

(IN) Specifies the name space of the *dirHandle/path* combination.

newDirHandle

(OUT) Points to the new directory handle.

newNameSpc

(IN) Specifies the name space to be used for the new directory handle.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89BF	INVALID_NAME_SPACE

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 87 06 Obtain File or Subdirectory Information
0x2222 87 12 Allocate Short Directory Handle

NWDeleteNSEntry

Erases the specified files from the file server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWDeleteNSEntry (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          fileName,
   nuint8          nameSpace,
   nuint16         searchAttr);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWDeleteNSEntry
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   fileName : pnstr8;
   nameSpace : nuint8;
   searchAttr : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare connection handle.

dirHandle

(IN) Specifies the directory handle on which files to be deleted currently reside.

fileName

(IN) Points to an absolute path (or relative if *dirHandle* is non-zero) that cannot exceed 255 characters in length.

nameSpace

(IN) Specifies the name space of *dirHandle/filePath*.

searchAttr

(IN) Specifies the file attributes to use in finding the file.

Return Values

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

0x0000	SUCCESSFUL
0x898A	NO_DELETE_PRIVILEGES
0x898D	SOME_FILES_AFFECTED_IN_USE
0x898E	NO_FILES_AFFECTED_IN_USE
0x898F	SOME_FILES_AFFECTED_READ_ONLY
0x8990	NO_FILES_AFFECTED_READ_ONLY
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	NO_FILES_FOUND_ERROR

Remarks

dirHandle must exist in the designated name space.

If a file has the immediate purge attribute set, the file cannot be recovered.

NCP Calls

0x2222 68 Erase File
0x2222 87 08 Delete A File Or Subdirectory

See Also

NWIntEraseFiles, **NWOpenCreateNSEntry**, **NWPurgeErasedFiles**, **NWRecoverDeletedFile**

NWGetDirectoryBase

Retrieves information used in further calls to the name space
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWGetDirectoryBase (
   NWCONN_HANDLE   conn,
   nuint8          dirHandle,
   pnstr8          path,
   nuint8          dstNamSpc,
   NW_IDX N_FAR   *idxStruct);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWGetDirectoryBase
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   path : pnstr8;
   dstNamSpc : nuint8;
   Var idxStruct : NW_IDX
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the directory to search.

path

(IN) Points to a valid DOS path (pointing to a directory or a file).

dstNamSpc

(IN) Specifies the destination name space.

idxStruct

(OUT) Points to NW_IDX.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89BF	INVALID_NAME_SPACE

Remarks

The *path* parameter must be upper case if the *dirHandle* parameter contains a DOS name space directory handle.

The *path* and *dirHandle* parameters must match the *dstNamSpc* parameter.

NetWare uses the *idxStruct* parameter as an index to quickly locate a directory entry (file or directory). It is required as a calling parameter to other functions and should not be modified by the application.

NCP Calls

0x2222 22 3 Get Directory Effective Rights
0x2222 22 19 Allocate Temporary Directory Handle
0x2222 22 20 Free Directory Handle
0x2222 23 15 Scan Files
0x2222 23 17 Get File Server Information
0x2222 68 File Erase
0x2222 87 2 Scan First
0x2222 87 3 Scan Next
0x2222 87 8 Delete Entry
0x2222 87 12 Allocate Directory Handle
0x2222 87 22 Generate Directory Base And Volume Number

See Also

NWNSGetMiscInfo, NWReadExtendedNSInfo, NWReadNSInfo , NWWriteExtendedNSInfo

File Service Group

NWGetHugeNSInfo

Gets extended (huge) NS information for the entry specified by *volNum, nameSpace* and *dirBase*

Local Servers: blocking

Remote Servers: blocking

Classification: 3.12, 4.x

SMP Aware: No

Service: Name Space

Syntax

```
#include <nwfile.h>

int NWGetHugeNSInfo (
   BYTE   volNum,
   BYTE   nameSpace,
   LONG   dirBase,
   LONG   hugeInfoMask,
   BYTE   *hugeStateInfo,
   BYTE   *hugeData,
   LONG   *hugeDataLen,
   BYTE   *nextHugeStateInfo);
```

Parameters

volNum

(IN) Volume number for which huge NS information is to be obtained.

nameSpace

(IN) Name space for which huge information is being returned.

dirBase

(IN) Directory base (or number) for the entry for which information is being obtained.

hugeInfoMask

(IN) Bit map that indicates which types of information the user wants returned. (Corresponds to the *extendedBitMask* in the NW_NS_INFO struct that can be retrieved by calling **NWQueryNSInfoFormat**.)

hugeStateInfo

(IN) The first time calling this function, this should be set to zeroes. On succeeding calls, the *nextHugeStateInfo* should be passed in this parameter.

hugeData

(OUT) Data returned as specified in the *hugeInfoMask*.

hugeDataLen

(OUT) Length of the huge data the name space returned.

nextHugeStateInfo

(OUT) Huge state information that should be passed in on the next call to this function. It is zero-filled when reading is done.

Return Values

ESuccess or NetWare errors

Remarks

This function retrieves extended NS information for *nameSpace* and returns it in *hugeData*.

See Also

NWGetDirBaseFromPath, **NWQueryNSInfoFormat**, **NWSetHugeNSInfo**

NWGetLongName

Retrieves a file name for *dstNamSpace*
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWGetLongName (
   NWCONN_HANDLE   conn,
   nuint8          dirHandle,
   pnstr8          path,
   nuint8          srcNamSpc,
   nuint8          dstNamSpc,
   pnstr8          longName);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWGetLongName
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   path : pnstr8;
   srcNamSpc : nuint8;
   dstNamSpc : nuint8;
   longName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the directory to scan.

path

(IN) Points to a valid path.

srcNamSpc

(IN) Specifies the name space referred to by *dirHandle/path*.

dstNamSpc

(IN) Specifies the name space for the return name.

longName

(OUT) Points to a buffer returning the corresponding name space's name (up to 256 bytes).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

For OS/2, the *dirHandle/path* combination must be in the same name space as *srcNamSpc*.

path can either be a fully specified path (vol:path), or it can be relative to *dirHandle*.

longName includes only the name of the last component in the path. **NWGetLongName** does not translate the entire path to a new name in the designated name space.

The name returned is the same name returned by **NWGetNSEntryInfo**.

NCP Calls

0x2222 87 06 Obtain File or Subdirectory Information

See Also

NWGetNSEntryInfo, **NWGetNSPath**, **NWSetLongName**

NWGetNameSpaceEntryName

Returns the name of a file or directory in the specified name space
Local Servers: blocking
Remote Servers: blocking
Classification: 3.12, 4.x
SMP Aware: No
Service: Name Space

Syntax

```
#include <nwfile.h>

int NWGetNameSpaceEntryName (
   BYTE   *path,
   LONG   nameSpace,
   LONG   maxNameBufferlength,
   BYTE   *nameSpaceEntryName);
```

Parameters

path

(IN) Specifies the path to the file system entry to get a name space entry name.

nameSpace

(IN) Specifies the name space to get the file or directory name for.

maxNameBufferLength

(IN) Specifies the maximum length of a name that can be stored in the buffer specified by *nameSpaceEntryName*.

nameSpaceEntryName

(IN) Points to a buffer in which to store the name.

Return Values

ESuccess or NetWare errors

Remarks

If you know the name of a file or directory in one name space---DOS, Macintosh, NFS---you can find out its name in other name spaces by calling **NWGetNameSpaceEntryName**.

See Also

File Service Group

NWSetNameSpaceEntryName

NWGetNSEntryInfo

Returns name space entry information for the entry referred to by the *dirHandle* and *path* combination

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWGetNSEntryInfo (
   NWCONN_HANDLE           conn,
   NWDIR_HANDLE            dirHandle,
   pnstr8                  path,
   nuint8                  srcNamSpc,
   nuint8                  dstNamSpc,
   nuint16                 searchAttrs,
   nuint32                 retInfoMask,
   NW_ENTRY_INFO N_FAR  *entryInfo);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWGetNSEntryInfo
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   path : pnstr8;
   srcNamSpc : nuint8;
   dstNamSpc : nuint8;
   searchAttrs : nuint16;
   retInfoMask : nuint32;
   Var entryInfo : NW_ENTRY_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired name

space (optional).

path

(IN) Points to the valid DOS path (pointing to a directory or file).

srcNamSpc

(IN) Specifies the name space of *dirHandle/path*.

dstNamSpc

(IN) Specifies the name space for the return information.

searchAttrs

(IN) Specifies the search attributes to use.

retInfoMask

(IN) Specifies the information to return.

entryInfo

(OUT) Points to NW_ENTRY_INFO. Only fields related to *retInfoMask* are valid.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89BF	INVALID_NAME_SPACE
0x89FF	Bad Parameter---no constant

Remarks

dirHandle can be zero if *path* contains the complete path, including the volume name. *dirHandle* and/or *path* contains the entry name according to *srcNamSpc*. This information is returned for *dstNamSpc*.

searchAttrs values follow:

C Value	Pascal Value	Value Name	Value Description

0x0000	$00	TA_NONE	Specifies no Reads or Writes are allowed.
0x0001	$01	TA_READ	Specifies file Reads are allowed.
0x0002	$02	TA_WRITE	Specifies file Writes are allowed.
0x0008	$08	TA_CREATE	Specifies files can be created.
0x0010	$10	TA_DELETE	Specifies files can be deleted.
0x0020	$20	TA_OWNERSHIP	Specifies subdirectories can be created or deleted and trustee rights granted or revoked.
0x0040	$40	TA_SEARCH	Specifies the directory can be searched.
0x0080	$80	TA_MODIFY	Specifies file attributes can be modified.
0x00FB	$FB	TA_ALL	Specifies the trustee has all the above rights to the directory.

To request information from a server, a client sets the appropriate bit or bits of *retInfoMask* and sends a request packet to the server.

NCP Calls

0x2222 87 06 Obtain File Or Subdirectory Information

See Also

NWGetOwningNameSpace, **NWGetLongName**

NWGetNSFileDirEntryNumber

Returns file information for a specified file under DOS and the name space associated with the specified file handle

NetWare Server: 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

N_EXTERN_LIBRARY (NWCCODE) NWGetNSFileDirEntryNumber (
   NWFILE_HANDLE   fileHandle,
   nuint8          nameSpace,
   pnuint32        volumeNum,
   pnuint32        directoryEntry,
   pnuint32        dataStream);
```

Pascal Syntax

```
Function NWGetNSFileDirEntryNumber
  (fileHandle : NWFILE_HANDLE;
   nameSpace : nuint8;
   volumeNum : pnuint32;
   directoryEntry : pnuint32;
   dataStream : pnuint32;
) : NWCCODE;
```

Parameters

fileHandle

(IN) Specifies the file handle.

nameSpace

(IN) Specifies the name space associated with the *directoryEntry* parameter.

volumeNum

(OUT) Points to the volume number of the file handle.

directoryEntry

(OUT) Points to the directory entry number in the name space associated with the *nameSpace* parameter.

dataStream

(OUT) Points to the data stream number if the name space is

(OUT) Points to the data stream number if the name space is NW_NS_MAC:

1 Data fork
0 Resource fork and anything else

Return Values

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

0x0000	SUCCESSFUL
0x0006	INVALID_HANDLE
0x8801	INVALID_CONNECTION
0x8988	INVALID_FILE_HANDLE

Remarks

NWGetNSFileDirEntryNumber returns the volume number and directory entry numbers in the name space specified by the *nameSpace* parameter.

Call the **NWGetFileDirEntryNumber** function to return the parent directory number. The **NWGetFileDirEntryNumber** allows you to specify the name space in which to return the parent directory number.

One way to create the file handle is to call the **NWOpenNSEntry** function. If you specify a long file name, the created file handle will be associated with the LONG name space. If a DOS file name is specified, the created file handle will be associated with the DOS name space.

The *nameSpace* parameter can have the following values:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_LONG

NCP Calls

87 31 Get File Information

See Also

File Service Group

NWOpenNSEntry

NWGetNSInfo

Returns the NW_NS_INFO structure to be used in reading and writing information to the name space

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWGetNSInfo (
   NWCONN_HANDLE       conn,
   NW_IDX N_FAR       *idxStruct,
   NW_NS_INFO N_FAR   *NSInfo);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWGetNSInfo
  (conn : NWCONN_HANDLE;
   Var idxStruct : NW_IDX;
   Var NSInfo : NW_NS_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

idxStruct

(IN) Points to the NW_IDX structure.

NSInfo

(OUT) Points to the NW_NS_INFO structure.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

NW_IDX is returned by **NWNSGetMiscInfo** or **NWGetDirectoryBase**. The *dstNameSpace* parameter in each function obtains the Name Space information.

NSInfo is returned for the destination name space in *idxStruct*.

NCP Calls

0x2222 87 23 Query NS Information Format

See Also

NWGetDirectoryBase, **NWNSGetMiscInfo**, **NWReadExtendedNSInfo**, **NWReadNSInfo**, **NWWriteExtendedNSInfo**, **NWWriteNSInfo**

NWGetNSInfo (NLM)

Returns specific NS information for the entry specified by the *volNum, nameSpace* and *dirBase* parameters

Local Servers: blocking

Remote Servers: blocking

Classification: 3.12, 4.x

SMP Aware: No

Service: Name Space

Syntax

```
#include <nwfile.h>

int NWGetNSInfo (
   BYTE   volNum,
   BYTE   srcNameSpace,
   BYTE   dstNameSpace,
   LONG   dirBase,
   LONG   nsInfoMask,
   BYTE  *nsSpecificInfo);
```

Parameters

volNum

(IN) Volume number for which information is to be returned.

srcNameSpace

(IN) Name space that corresponds with the *dirBase* being passed.

dstNameSpace

(IN) Name space in which the information is to be returned.

dirBase

(IN) Directory base (or number) for the entry for which information is being retrieved.

nsInfoMask

(IN) Bit map that indicates which types of information the user wants returned in the data parameter.

nsSpecificInfo

(OUT) Data that was asked for as indicated in the *nsInfoMask*.

Return Values

ESuccess or NetWare errors

Remarks

If the current name space is NFS, a value of 2 (for NFS) would be passed to the *srcNameSpace* parameter. However, if the returned information should be in the Macintosh name space format, a value of 1 would be passed to the *dstNameSpace* parameter.

See DOS Name Space Bit Mask.

See Also

NWGetDirBaseFromPath, **NWQueryNSInfoFormat**, **NWSetNSInfo**

NWGetNSLoadedList

Retrieves a list of the name spaces loaded for the specified volume
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWGetNSLoadedList (
   NWCONN_HANDLE   conn,
   nuint8          volNum,
   nuint8          maxListLen,
   pnuint8         NSLoadedList,
   pnuint8         actualListLen);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWGetNSLoadedList
  (conn : NWCONN_HANDLE;
   volNum : nuint8;
   maxListLen : nuint8;
   NSLoadedList : pnuint8;
   actualListLen : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number to obtain the list from.

maxListLen

(IN) Specifies the size of *NSLoadedList* (in bytes).

NSLoadedList

(OUT) Points to a buffer (*maxListLen* bytes).

actualListLen

actualListLen

(OUT) Points to the number of name spaces loaded (in bytes).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

NSLoadedList contains a nuint8 entry for every name space loaded on the server. The buffer for *NSLoadedList* should be at least 5 bytes long (*maxListLen* should also be at least 5 bytes).

NCP Calls

0x2222 87 24 Get Name Spaces Loaded List From Volume Number

NWGetNSLoadedList (NLM)

Retrieves a list of the name spaces that are loaded on the specified volume
Local Servers: blocking
Remote Servers: blocking
Classification: 3.12, 4.x
SMP Aware: No
Service: Name Space

Syntax

```
#include <nwfile.h>

int NWGetNSLoadedList (
   BYTE   volNum,
   WORD   loadListSize,
   BYTE   *NSLoadedList,
   WORD   *returnListSize);
```

Parameters

volNum

(IN) Volume number for which to get the list of loaded name spaces.

loadListSize

(IN) Size (in bytes) of the *NSLoadedList* buffer being passed.

NSLoadedList

(OUT) Pointer to a buffer to hold the loaded name spaces.

returnListSize

(OUT) Pointer to the number of name spaces loaded.

Return Values

ESuccess or NetWare errors

Remarks

The *NSLoadedList* contains a BYTE entry for every name space that is loaded on the volume. The buffer for *NSLoadedList* needs to be at least MAX_NAMESPACES bytes long (therefore, *loadListSize* needs to be at least MAX_NAMESPACES). In the case where there are more name spaces loaded than there is space available in the *NSLoadedList* buffer, *returnListSize* contains the number of name spaces loaded.

See Also

File Service Group

NWQueryNSInfoFormat

NWGetNSPath

Returns the full NetWare path for the desired name space associated with the specified path

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWGetNSPath (
   NWCONN_HANDLE       conn,
   nuint8              dirHandle,
   nuint16             fileFlag,
   nuint8              srcNamSpc,
   nuint8              dstNamSpc,
   NW_NS_PATH N_FAR  *NSPath);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWGetNSPath
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   fileFlag : nuint16;
   srcNamSpc : nuint8;
   dstNamSpc : nuint8;
   Var NSPath : NW_NS_PATH
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired name space.

fileFlag

(IN) Specifies whether the source path ends with a file or a directory

name:

```
0 = directory name
1 = file name
```

srcNamSpc

(IN) Specifies the name space used for *srcPath* in *NSPath*.

dstNamSpc

(IN) Specifies the name space for the return path.

NSPath

(IN/OUT) Points to NW_NS_PATH.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

A full path includes the volume name. For example:

```
volume:path\path
```

If the *fileFlag* parameter is set to 0 (indicating a directory name is being passed) and a file name is passed, INVALID_PARAMETER will be returned. The same error will be returned if the *fileFlag* parameter is set to 1 (indicating a file name is being passed) and a directory name is passed.

On NetWare server versions 3.12 and before, **NWGetNSPath** will return INVALID_PATH when used to return the full path of a root file.

NCP Calls

0x2222 87 28 Get Full Path String

NWGetOwningNameSpace

Returns the owning name space for the specified directory or file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWGetOwningNameSpace (
   NWCONN_HANDLE   conn,
   nuint8          dirHandle,
   pnstr8          path,
   pnuint8         namSpc);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWGetOwningNameSpace
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   path : pnstr8;
   namSpc : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the directory to search.

path

(IN) Points to a valid NetWare path (pointing to a directory or file).

namSpc

(OUT) Points to the owning name space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

The owning name space is defined as the name space under which the entry (file or directory) was created.

Both the *dirHandle* and *path* parameters must be in the default name space.

The default name space is the name space that matches the OS and the loaded name spaces on that volume. For example, Windows95 on a volume with OS/2 (LONG) name space will set OS/2 (LONG) name space as the default name space.

NCP Calls

0x2222 87 06 Obtain File or Subdirectory Information

NWIsLNSSupportedOnVolume

Queries the NetWare server and returns a nonzero if the OS/2 name space is supported on the target volume

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWIsLNSSupportedOnVolume (
   NWCONN_HANDLE    conn,
   NWDIR_HANDLE     dirHandle,
   pnstr8           path);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWIsLNSSupportedOnVolume
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the volume whose status is being checked.

path

(IN) Points to the absolute directory path (or a path relative to the directory handle) associated with the volume whose status is being checked.

Return Values

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

0x0000	OS/2 name space not supported on volume
nonzero	OS/2 name space supported on volume

Remarks

NWIsLNSSupportedOnVolume is called in OS/2 to determine whether DOS names or HPFS names should be used in paths.

In DOS and Windows, 0x0000 will always be returned.

In OS/2, if a nonzero value is returned, use HPFS names when calling NWCalls. On 3.11 servers and above, NWCalls expect HPFS names to be used on all volumes having the OS/2 name space loaded.

In OS/2, if the *dirHandle* or *path* parameters are invalid, 0x0000 will always be returned. Therefore, make sure the *dirHandle* and *path* parameters are valid before calling **NWIsLNSSupportedOnVolume**.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 234 Get Connection's Task Information

NWNSGetDefaultNS

Returns the default name space
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWNSGetDefaultNS (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   pnuint8         pbuDefaultNameSpace);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWNSGetDefaultNS
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   pbuDefaultNameSpace : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the directory for which to return the default name space.

path

(IN) Points to a valid NetWare path (pointing to a directory or a file).

pbuDefaultNameSpace

(OUT) Points to the default name space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8836	INVALID_PARAMETER
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x89FF	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

Both the *dirHandle* and *path* parameters must be in the default name space.

The default name space is the name space that matches the OS and the loaded name spaces on that volume. For example, Windows95 on a volume with OS/2 (LONG) name space will set OS/2 (LONG) name space as the default name space.

NCP Calls

0x2222 22 5 Get Volume Number
0x2222 22 21 Get Volume Info With Handle
0x2222 87 24 Get Name Spaces Loaded List From Volume Number

See Also

NWGetVolumeInfoWithHandle, **NWGetVolumeNumber**

NWNSGetMiscInfo

Retrieves information to be used in further calls to the name space
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWNSGetMiscInfo (
   NWCONN_HANDLE   conn,
   nuint8          dirHandle,
   pnstr8          path,
   nuint8          dstNameSpace,
   NW_IDX N_FAR   *idxStruct);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWNSGetMiscInfo
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   path : pnstr8;
   dstNameSpace : nuint8;
   Var idxStruct : NW_IDX
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the directory to search.

path

(IN) Points to a valid NetWare path (pointing to a directory or a file).

dstNameSpace

(IN) Specifies the destination name space.

idxStruct

(OUT) Points to NW_IDX.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89BF	INVALID_NAME_SPACE

Remarks

dirHandle/path should match *dstNameSpace*.

Both the *dirHandle* and *path* parameters must be in the default name space.

The default name space is the name space that matches the OS and the loaded name spaces on that volume. For example, Windows95 on a volume with OS/2 (LONG) name space will set OS/2 (LONG) name space as the default name space.

NetWare uses NW_IDX as an index to quickly locate a directory entry (file or directory). NW_IDX is required as a parameter for other functions and should not be modified by the application.

NCP Calls

0x2222 87 06 Obtain File or Subdirectory Information

See Also

NWGetDirectoryBase

NWNSRename

Renames an entry in the specified name space, given a path specifying the entry name

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWNSRename (
   NWCONN_HANDLE   conn,
   nuint8          dirHandle,
   nuint8          namSpc,
   pnstr8          oldName,
   nuint16         oldType,
   pnstr8          newName,
   nuint8          renameFlag);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWNSRename
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   namSpc : nuint8;
   oldName : pnstr8;
   oldType : nuint16;
   newName : pnstr8;
   renameFlag : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle of the parent directory.

namSpc

(IN) Specifies the name space of *oldName*.

oldName

(IN) Points to the name of the directory or file to rename.

oldType

(IN) Specifies the type of *oldName*.

newName

(IN) Points to the new name (256 bytes maximum).

renameFlag

(IN) Specifies whether name conversion should be done; ignored for NetWare 3.11 and below.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899E	INVALID_FILENAME

Remarks

A transaction file cannot be deleted or renamed.

dirHandle must point to the parent directory.

oldName and *newName* must be valid names containing only one component. *dirHandle* will specify the path.

oldType can be one of the following values:

C Value	Pascal Value	Value Name
0x8000	$0800	NW_TYPE_FILE
0x0010	$0010	NW_TYPE_SUBDIR

renameFlag can be one of the following:

C Value	Pascal Value	Value Name
0x03	$03	NW_NAME_CONVERT
0x04	$04	NW_NO_NAME_CONVERT

The default operation for **NWNSRename** is to rename the file in all name spaces, report an error if renaming a file as itself, and do nothing with the file compatibility mode. When NW_NAME_CONVERT is passed in the *renameFlag* parameter, renaming the file to the same name will not report an error and compatibility mode will be set for that file. If NW_NO_NAME_CONVERT is passed in *renameFlag*, the new name is changed only in the specified name space. When renaming is done the shortening algorithm is used for the DOS and/or MAC name spaces when necessary.

AFP directory and file names (long names) contain 1-31 characters. A long name is a Pascal string preceded by one byte which specifies the length of the name. Long names can contain any ASCII character between 1 and 255 except the colon (:) but cannot be terminated by a NULL character (character 0).

The NetWare server automatically generates DOS-style file names (short names) for all AFP directories, as well as for created files and accessed files. The NetWare server maintains both the long name and the short name for each AFP directory and file.

NetWare uses the following conventions to convert AFP names to DOS names:

If a long name containing no periods is converted to a short name, the first eight valid DOS characters of the long name are used:

```
Long Name:      THIS IS A NAME
Short Name:     THISISAN
```

If a long name contains a period within the first nine valid DOS characters, the first eight characters before the period and the first three characters after the last period are used:

```
Long Name:      THIS.IS.A.NAME
Short Name:     THIS.NAM
```

If an application creates in the same directory two files whose initial 8 short name characters are the same, the NetWare server replaces the last character of the second file's short name with an ascending

decimal integer guaranteeing its uniqueness:

Example 1

```
Long Name:      THIS IS THE FIRST FILE
Short Name:     THISISTH
Long Name:      THIS IS THE SECOND FILE
Short Name:     THISIST1
```

Example 2

```
Long Name:      THIS IS A 1 TIME OFFER
Short Name:     THISISA1
Long Name:      THIS IS A 1 TIME DEAL
Short Name:     THISISA2
```

NOTE: If the first file in Example 1 is subsequently deleted, a third file whose DOS name would have been identical to the first and second names is created in that directory. The third name is identical to the deleted first name and will not be appended with a decimal integer.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 87 04 Rename Or Move A File Or Subdirectory

See Also

NWGetLongName

NWOpenCreateNSEntry

Opens a file in the specified name space or creates and then opens a file if it does not already exist

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWOpenCreateNSEntry (
   NWCONN_HANDLE             conn,
   nuint8                    dirHandle,
   nuint8                    namSpc,
   pnstr8                    path,
   NW_NS_OPENCREATE N_FAR   *NSOpenCreate,
   NWFILE_HANDLE N_FAR     *fileHandle);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWOpenCreateNSEntry
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   namSpc : nuint8;
   path : pnstr8;
   Var NSOpenCreate : NW_NS_OPENCREATE;
   Var fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare connection handle.

dirHandle

(IN) Specifies the directory handle on which to open/create the specified file.

namSpc

(IN) Specifies the name space of *dirHandle/path*.

path

(IN) Points to an absolute path, (or relative if *dirHandle* is nonzero).

NSOpenCreate

(IN/OUT) Points to NW_NS_OPENCREATE containing information needed to create the entry on input. Points to NW_NS_OPENCREATE containing the results of a successful open/create upon output

fileHandle

(OUT) Points to the NWFILE_HANDLE. When you are creating subdirectories, *fileHandle* returns zero.

Return Values

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

0x0000	SUCCESSFUL
0x8980	ERR_LOCK_FAIL
0x8981	NO_MORE_FILE_HANDLES
0x8982	NO_OPEN_PRIVILEGES
0x8994	NO_WRITE_PRIVILEGES_OR_READONLY
0x8996	SERVER_OUT_OF_MEMORY
0x8998	SERVER_DOES_NOT_EXIST
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	Failure

NCP Calls

0x2222 23 17 Get File Server Info
0x2222 66 File Close
0x2222 87 1 Open/Create Entry
0x2222 87 30 Open/Create File or Subdirectory

See Also

NWDeleteNSEntry

NWOpenDataStream

Opens a data stream associated with any supported name space on the server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWOpenDataStream (
   NWCONN_HANDLE          conn,
   nuint8                 dirHandle,
   pnstr8                 fileName,
   nuint16                dataStream,
   nuint16                attrs,
   nuint16                accessMode,
   pnuint32               NWHandle,
   NWFILE_HANDLE N_FAR  *fileHandle);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWOpenDataStream
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   fileName : pnstr8;
   dataStream : nuint16;
   attrs : nuint16;
   accessMode : nuint16;
   NWHandle : pnuint32;
   Var fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the directory

containing the file.

fileName

(IN) Points to the name of the file containing the data stream.

dataStream

(IN) Specifies the data stream number if the name space is Mac OS:

0 =Resource Fork
1=Data Fork

For DOS, always pass 0.

attrs

(IN) Specifies the attributes to use in searching for the file to open.

accessMode

(IN) Specifies the rights to use in opening the file.

NWHandle

(OUT) Points to a 4-byte NetWare handle to *dataStream* (optional).

fileHandle

(OUT) Points to a file handle.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x8980	ERR_LOCK_FAIL
0x8982	NO_OPEN_PRIVILEGES
0x8990	NO_FILES_AFFECTED_READ_ONLY
0x89BE	INVALID_DATA_STREAM
0x89FF	NO_FILES_FOUND_ERROR

Remarks

NWOpenDataStream also obtains a NetWare file handle to a data stream.

Three data streams are defined for NetWare:

NW_DS_DOS 0
NW_DS_MAC 1
NW_DS_FTAM 2

All name spaces can access their main data stream through the DOS data stream. For example, the Macintosh data fork can be accessed through this data stream. However, it would be impractical to access the data for the DOS data stream through **NWOpenDataStream** since that stream is available through the open functions for the particular client OS.

These constants identify trustee access rights for opening a a directory with **NWOpenDataStream**.

C Value	Pascal Value	Value Name	Value Description
0x00	$00	TA_NONE	Specifies no Reads or Writes are allowed.
0x01	$01	TA_READ	Specifies file Reads are allowed.
0x02	$02	TA_WRITE	Specifies file Writes are allowed.
0x08	$08	TA_CREATE	Specifies files can be created.
0x10	$10	TA_DELETE	Specifies files can be deleted.
0x20	$20	TA_OWNERSHIP	Specifies subdirectories can be created or deleted and trustee rights granted or revoked.
0x40	$40	TA_SEARCH	Specifies the directory can be searched.
0x80	$80	TA_MODIFY	Specifies file attributes can be modified.
0xFB	$FB	TA_ALL	Specifies the trustee has all the above rights to the directory.

attrs definitions follow:

C Value	Pascal Value	Value Name
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM

0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

accessMode definitions follow:

C Value	Pascal Value	Value Name
0x0001	$0001	AR_READ
0x0002	$0002	AR_WRITE
0x0001	$0001	AR_READ_ONLY
0x0002	$0002	AR_WRITE_ONLY
0x0004	$0004	AR_DENY_READ
0x0008	$0008	AR_DENY_WRITE
0x0010	$0010	AR_COMPATABILITY
0x0040	$0040	AR_WRITE_THROUGH
0x0100	$0100	AR_OPEN_COMPRESSED

For DOS, the NetWare shell currently offers a function to get a DOS handle from a NetWare handle.

NCP Calls

0x2222 22 49 Open Data Stream
0x2222 66 File Close
0x2222 87 06 Obtain File or Subdirectory Information

See Also

NWAFPOpenFileFork, NWConvertHandle

NWOpenNSEntry

Opens or creates a file or creates a subdirectory with a given owning name space

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWOpenNSEntry (
   NWCONN_HANDLE           conn,
   nuint8                  dirHandle,
   nuint8                  namSpc,
   nuint8                  dataStream,
   pnstr8                  path,
   NW_NS_OPENCREATE N_FAR       *NSOpen,
   NWFILE_HANDLE N_FAR    *fileHandle);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWOpenNSEntry
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   namSpc : nuint8;
   dataStream : nuint8;
   path : pnstr8;
   Var NSOpen : NW_NS_OPEN;
   Var fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the directory in which to create the file.

namSpc

(IN) Specifies the name space for the file creation.

dataStream

(IN) Specifies the data stream number if the name space is Mac OS:

0 =Resource Fork
1=Data Fork

For DOS, always pass 0.

path

(IN) Points to the name to use in creating the file. Optionally contains a volume:path specification.

NSOpen

(IN/OUT) Points to NW_NS_OPENCREATE containing the information needed to open the entry. Results of a successful open are also returned in NW_NS_OPENCREATE.

fileHandle

(OUT) Points to the OS file handle; it returns zero if you are creating subdirectories.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

dirHandle can be zero if the path contains the complete path, including the volume name. (*dirHandle*/*path* should match *namSpc*.)

OC_MODE_ constants used in *openCreateMode* are listed below:

C Value	**Pascal Value**	**Value Name**
0x01	$01	OC_MODE_OPEN

0x02	$02	OC_MODE_TRUNCATE
0x02	$02	OC_MODE_REPLACE
0x08	$08	OC_MODE_CREATE

SA_ constants used in *searchAttributes* are listed below:

C Value	**Pascal Value**	**Value Name**
0x0000	$0000	SA_NORMAL
0x0002	$0002	SA_HIDDEN
0x0004	$0004	SA_SYSTEM
0x0010	$0010	SA_SUBDIR_ONLY
0x8000	$8000	SA_SUBDIR_FILES
0x8006	$8006	SA_ALL

AR_ constants used in *desiredAccessRights* are listed below:

C Value	**Pascal Value**	**Value Name**
0x0001	$0001	AR_READ
0x0002	$0002	AR_WRITE
0x0001	$0001	AR_READ_ONLY
0x0002	$0002	AR_WRITE_ONLY
0x0004	$0004	AR_DENY_READ
0x0008	$0008	AR_DENY_WRITE
0x0010	$0010	AR_COMPATABILITY

0x0040	$0040	AR_WRITE_THROUGH
0x0100	$0100	AR_OPEN_COMPRESSED

OC_ACTION_ constants used in *openCreateAction* are listed below:

C Value	Pascal Value	Value Name
0x01	$01	OC_ACTION_NONE
0x01	$01	OC_ACTION_OPEN
0x02	$02	OC_ACTION_CREATE
0x04	$04	OC_ACTION_TRUNCATE
0x04	$04	OC_ACTION_REPLACE

The file handle returned is appropriate for the platform the API is written for. This file handle may be used for access to the attribute value through standard file I/O with the handle. This includes closing the file as well as reading and writing to the file.

For Windows, call **_lread**, **_lwrite**, **_lclose**, and **_lseek** rather than calling the standard file I/O functions. Calling standard file I/O functions in Windows returns unexpected results.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 66 File Close
0x2222 87 01 Open Create File Or Subdirectory
0x2222 87 30 Open/Create File Or Subdirectory

NWQueryNSInfoFormat

Returns the NW_NS_INFO structure to be used in getting and setting name space information

Local Servers: blocking

Remote Servers: blocking

Classification: 3.12, 4.x

SMP Aware: No

Service: Name Space

Syntax

```
#include <nwfile.h>

int NWQueryNSInfoFormat (
   BYTE          nameSpace,
   BYTE          volNum,
   NW_NS_INFO    *nsInfo);
```

Parameters

nameSpace

(IN) Specifies the name space to return information for.

volNum

(IN) Specifies the volume number to return information for.

nsInfo

(OUT) Points to an NW_NS_INFO structure.

Return Values

ESuccess or NetWare errors

Remarks

The *nsInfo* parameter points to an NW_NS_INFO structure. This structure is defined in nwfile.h as follows:

```
typedef struct
{
   LONG   nsInfoBitMask;
   LONG   fixedBitMask;
   LONG   reservedBitMask;
   LONG   extendedBitMask;
   WORD   fixedBitsDefined;
   WORD   reservedBitsDefined;
```

```
    WORD   extendedBitsDefined;
    LONG   fieldsLenTable[32];
    BYTE   hugeStateInfo[16];
    LONG   hugeDataLength;
} NW_NS_INFO;
```

See Also

NWGetNSInfo (NLM), NWSetNSInfo

NWReadExtendedNSInfo

Reads the extended (huge) name space information for the specified name space

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWReadExtendedNSInfo (
   NWCONN_HANDLE       conn,
   NW_IDX N_FAR       *idxStruct,
   NW_NS_INFO N_FAR   *NSInfo,
   pnuint8            data);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWReadExtendedNSInfo
  (conn : NWCONN_HANDLE;
   Var idxStruct : NW_IDX;
   Var NSInfo : NW_NS_INFO;
   data : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

idxStruct

(IN) Points to NW_IDX returned from **NWNSGetMiscInfo**.

NSInfo

(IN) Points to NW_NS_INFO returned from **NWGetNSInfo**.

data

(OUT) Points to a buffer containing the data from the name space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

If *extendedBitMask* is set in NW_NS_INFO, **NWReadExtendedNSInfo** should be used to read the extended information. *extendedBitMask* contains a Read-only information field that should be preserved. The application must not manipulate *extendedBitMask*; it must not be zero.

dstNameSpace and *dstDirBase* of NW_IDX are used to determine the target name space of **NWReadExtendedNSInfo**.

NCP Calls

0x2222 87 26 Get Huge NS Information

See Also

NWGetDirectoryBase, **NWGetNSInfo**, **NWNSGetMiscInfo**, **NWWriteExtendedNSInfo**

NWReadNSInfo

Reads name space information from the designated name space
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWReadNSInfo (
   NWCONN_HANDLE       conn,
   NW_IDX N_FAR       *idxStruct,
   NW_NS_INFO N_FAR   *NSInfo,
   pnuint8            data);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWReadNSInfo
  (conn : NWCONN_HANDLE;
   Var idxStruct : NW_IDX;
   Var NSInfo : NW_NS_INFO;
   data : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

idxStruct

(IN) Points to NW_IDX returned from **NWNSGetMiscInfo**.

NSInfo

(IN) Points to NW_NS_INFO returned from **NWGetNSInfo**.

data

(OUT) Points to a 512-byte buffer receiving data from the name space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

NSInfoBitMask bit definitions follow:

C Value	Pascal Value	Value Name
0x0002L	$0002	DM_ATTRIBUTES
0x0004L	$0004	DM_CREATE_DATE
0x0008L	$0008	DM_CREATE_TIME
0x0010L	$0010	DM_CREATOR_ID
0x0020L	$0020	DM_ARCHIVE_DATE
0x0040L	$0040	DM_ARCHIVE_TIME
0x0080L	$0080	DM_ARCHIVER_ID
0x0100L	$0100	DM_MODIFY_DATE
0x0200L	$0200	DM_MODIFY_TIME
0x0400L	$0400	DM_MODIFIER_ID
0x0800L	$0800	DM_LAST_ACCESS_DATE
0x1000L	$1000	DM_INHERITED_RIGHTS_MASK
0x2000L	$2000	DM_MAXIMUM_SPACE

NCP Calls

0x2222 87 19 Get NS Information

See Also

NWGetNSEntryInfo, NWWriteNSInfo

NWScanNSEntryInfo

Obtains directory entry information using a specific name space
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWScanNSEntryInfo (
   NWCONN_HANDLE           conn,
   nuint8                  dirHandle,
   nuint8                  namSpc,
   nuint16                 attrs,
   SEARCH_SEQUENCE N_FAR  *sequence,
   pnstr8                  srchPattern,
   nuint32                 retInfoMask,
   NW_ENTRY_INFO N_FAR    *entryInfo);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWScanNSEntryInfo
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   namSpc : nuint8;
   attrs : nuint16;
   Var sequence : SEARCH_SEQUENCE;
   searchPattern : pnstr8;
   retInfoMask : nuint32;
   Var entryInfo : NW_ENTRY_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the directory to scan. Must point to the parent directory.

namSpc

(IN) Specifies the name space of *dirHandle*.

attr

(IN) Specifies the attributes to be used for the scan.

sequence

(IN/OUT) Points to SEARCH_SEQUENCE.

srchPattern

(IN) Points to the name of the entry for which to scan (wildcards are allowed).

retInfoMask

(IN) Specifies the information to return.

entryInfo

(OUT) Points to NW_ENTRY_INFO.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

NWScanNSEntryInfo can be used iteratively with wild cards. On the first call, *searchDirNumber* in the SEARCH_SEQUENCE structure should be set to -1. After that, the server manages the information.

retInfoMask is used to determine which fields of NW_ENTRY_INFO to return. *nameLength* and *entryName* are always returned in **NWScanNSEntryInfo**.

To request information from a server, a client sets the appropriate bit or bits of *retInfoMask* and sends a request packet to the server. *retInfoMask* values follow:

C Value	Pascal Value	Value Name

0x0001L	$0001	IM_NAME
0x0001L	$0001	IM_ENTRY_NAME
0x0002L	$0002	IM_SPACE_ALLOCATED
0x0004L	$0004	IM_ATTRIBUTES
0x0008L	$0008	IM_SIZE
0x0010L	$0010	IM_TOTAL_SIZE
0x0020L	$0020	IM_EA
0x0040L	$0040	IM_ARCHIVE
0x0080L	$0080	IM_MODIFY
0x0100L	$0100	IM_CREATION
0x0200L	$0200	IM_OWNING_NAMESPACE
0x0400L	$0400	IM_DIRECTORY
0x0800L	$0800	IM_RIGHTS
0x0FEDL	$0FED	IM_ALMOST_ALL
0x0FFFL	$0FFF	IM_ALL
0x1000L	$1000	IM_REFERENCE_ID
0x2000L	$2000	IM_NS_ATTRIBUTES
0x4000L	$4000	IM_DATASTREAM_SIZES
0x80000000L	$80000000	IM_COMPRESSED_INFO
0x80000000L	$80000000	IM_NS_SPECIFIC_INFO

Possible *attr* values follow:

C Value	**Pascal Value**	**Value Name**
0x00	$00	FA_NORMAL
0x01	$01	FA_READ_ONLY
0x02	$02	FA_HIDDEN
0x04	$04	FA_SYSTEM
0x08	$08	FA_EXECUTE_ONLY
0x10	$10	FA_DIRECTORY
0x20	$20	FA_NEEDS_ARCHIVED
0x80	$80	FA_SHAREABLE

NCP Calls

0x2222 87 02 Initialize Search
0x2222 87 03 Search For File Or Subdirectory

See Also

NWGetNSEntryInfo

NWSetHugeNSInfo

Sets extended (huge) NS information for the entry specified by *volNum, nameSpace,* and *dirBase*

Local Servers: blocking

Remote Servers: blocking

Classification: 3.12*,* 4.x

SMP Aware: No

Service: Name Space

Syntax

```
#include <nwfile.h>

int NWSetHugeNSInfo (
   BYTE   volNum,
   BYTE   nameSpace,
   LONG   dirBase,
   LONG   hugeInfoMask,
   BYTE   *hugeStateInfo,
   LONG   *hugeDataLen,
   BYTE   *hugeData,
   BYTE   *nextHugeStateInfo,
   LONG   *hugeDataUsed);
```

Parameters

volNum

(IN) Volume number for which to set huge NS information.

nameSpace

(IN) Name space for which to set huge information.

dirBase

(IN) Directory base (or number) for the entry for which to set information.

hugeInfoMask

(IN) Bit map that indicates which types of information is being set.

hugeStateInfo

(IN) Information that helps the name space transfer the data across the wire. The *hugeStateInfo* is information that was returned by a previous call to **NWGetHugeNSInfo**.

hugeDataLen

(IN) Length of the huge data to be set.

hugeData

(IN) Data to be set as specified in the *hugeInfoMask*.

nextHugeStateInfo

(OUT) Huge state information that should be passed in on the next call to this function should all the information not fit in one packet.

hugeDataUsed

(OUT) Number of bytes that were actually set by the name space.

Return Values

ESuccess or NetWare errors

Remarks

This function sets extended NS information for an entry in the specified name space.

See Also

NWGetDirBaseFromPath, **NWGetHugeNSInfo**, **NWQueryNSInfoFormat**

NWSetLongName

Renames an entry in the specified name space, given a path specifying the entry name

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWSetLongName (
   NWCONN_HANDLE   conn,
   nuint8          dirHandle,
   nuint8          namSpc,
   pnstr8          dstPath,
   nuint16         dstType,
   pnstr8          longName);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWSetLongName
  (conn : NWCONN_HANDLE;
   dirHandle : nuint8;
   namSpc : nuint8;
   dstPath : pnstr8;
   dstType : nuint16;
   longName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle of the parent directory.

namSpc

(IN) Specifies the name space of *dstPath*.

dstPath

(IN) Points to the name of the directory or file to rename.

dstType

(IN) Specifies the directory or file type that *dstPath* points to.

longName

(IN) Points to the new name (256 bytes maximum).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x899E	INVALID_FILENAME

Remarks

dirHandle must point to the parent directory.

dstPath and *longName* must be valid names containing only one component. *dirHandle* will specify the path where the one component is located.

dstType can take on the following values:

C Value	Pascal Value	Value Name
0x8000	$0800	NW_TYPE_FILE
0x0010	$0010	NW_TYPE_SUBDIR

Resetting a filename in one name space resets the name in all name spaces. The shortening algorithm is used for the DOS and/or Macintosh name spaces, if appropriate.

AFP directory and file names (long names) contain from 1 to 31

characters. A long name is a Pascal string preceded by one byte which specifies the length of the name. Long names can contain any ASCII character between 1 and 255 except the colon (:) but cannot be terminated by a NULL character (character 0). NetWare servers automatically generate DOS-style file names (short names) for all AFP directories, as well as for created files and accessed files. NetWare servers maintain both the long name and the short name for each AFP directory and file.

NetWare uses the following conventions to convert AFP names to DOS names:

If a long name containing no periods is converted to a short name, the first eight valid DOS characters of the long name are used:

```
Long Name:    THIS IS A NAME
Short Name:   THISISAN
```

If a long name contains a period within the first nine valid DOS characters, the first eight characters before the period and the first three characters after the last period are used:

```
Long Name:    THIS.IS.A.NAME
Short Name:   THIS.NAM
```

Note that periods preceding file names are dropped:

```
long name:    this.is.name
short name:   this.name
```

If an application creates two files with identical short names (initial eight characters) in the same directory, the NetWare server replaces the last character of the second file's short name with an ascending decimal integer that will guarantee its uniqueness:

Example 1

```
Long Name:    THIS IS THE FIRST FILE
Short Name:   THISISTH
Long Name:    THIS IS THE SECOND FILE
Short Name:   THISIST1
```

Example 2

```
Long Name:    THIS IS A 1 TIME OFFER
Short Name:   THISISA1
Long Name:    THIS IS A 1 TIME DEAL
Short Name:   THISISA2
```

NOTE: If the first file in example 1 is subsequently deleted, a third file whose DOS name would have been identical to the first and second names is created in that directory. The third name is identical to the deleted first name and will not be appended with a decimal integer.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 87 04 Rename Or Move A File Or Subdirectory

See Also

NWGetLongName

NWSetNameSpaceEntryName

Sets the name of a file or directory in the specified name space
Local Servers: blocking
Remote Servers: blocking
Classification: 3.12, 4.x
SMP Aware: No
Service: Name Space

Syntax

```
#include <nwfile.h>

int NWSetNameSpaceEntryName (
   BYTE   *path,
   LONG   nameSpace,
   BYTE   *nameSpaceEntryName);
```

Parameters

path

(IN) Specifies the path of the file system entry to set a name space entry name for.

nameSpace

(IN) Specifies the name space to set the file or directory name for.

nameSpaceEntryName

(IN) Points to an ASCIIZ string that specifies the new file or directory name in the specified name space.

Return Values

ESuccess or NetWare errors

Remarks

This function sets the file system entry's name in the specified name space only. The naming change is not reflected in the other name space entries.

See Also

NWSetNameSpaceEntryName

Example

See the example for **NWGetNameSpaceEntryName**.

NWSetNSEntryDOSInfo

Modifies information in one name space using a path from another name space

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWSetNSEntryDOSInfo (
   (NWCONN_HANDLE             conn,
    NWDIR_HANDLE              dirHandle,
    pnstr8                    path,
    nuint8                    namSpc,
    nuint16                   searchAttrs,
    nuint32                   modifyDOSMask,
    MODIFY_DOS_INFO N_FAR  *dosInfo);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWSetNSEntryDOSInfo
   (conn : NWCONN_HANDLE;
    dirHandle : nuint8;
    path : pnstr8;
    namSpc : nuint8;
    searchAttrs : nuint16;
    modifyDOSMask : nuint32;
    Var dosInfo : MODIFY_DOS_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle of the parent directory.

path

(IN) Points to the path.

namSpc

(IN) Specifies the name space of *dirHandle* and *path*.

searchAttrs

(IN) Specifies the search attributes to use.

modifyDOSMask

(IN) Specifies the information to return.

dosInfo

(IN) Points to MODIFY_DOS_INFO containing the information specified by *luModifyDOSMask*.

Return Values

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

0x0000	SUCCESSFUL
0x89FF	NO_FILES_FOUND_ERROR

Remarks

suSrchAttr can have the following values:

C Value	Pascal Value	Value Name
0x0002	$0002	SA_HIDDEN
0x0004	$0004	SA_SYSTEM
0x0010	$0010	SA_SUBDIR_ONLY
0x8000	$8000	SA_SUBDIR_FILES

luModifyDOSMask can have the following values:

C Value	Pascal Value	Value Name

	Value	
0x0002 L	$0002	DM_ATTRIBUTES
0x0004 L	$0004	DM_CREATE_DATE
0x0008 L	$0008	DM_CREATE_TIME
0x0010 L	$0010	DM_CREATOR_ID
0x0020 L	$0020	DM_ARCHIVE_DATE
0x0040 L	$0040	DM_ARCHIVE_TIME
0x0080 L	$0080	DM_ARCHIVER_ID
0x0100 L	$0100	DM_MODIFY_DATE
0x0200 L	$0200	DM_MODIFY_TIME
0x0400 L	$0400	DM_MODIFIER_ID; cannot be set for subdirectories
0x0800 L	$0800	DM_LAST_ACCESS_DATE; cannot be set for subdirectories
0x1000 L	$1000	DM_INHERITED_RIGHTS_MASK
0x2000 L	$2000	DM_MAXIMUM_SPACE

DM_MODIFIER_ID and DM_LAST_ACCESS_DATE cannot be used when the *suSrchAttr* parameter contains SA_SUBDIR_ONLY. The server masks off DM_MODIFIER_ID and DM_LAST_ACCESS_DATE on subdirectories. If the resultant mask is 0x0000, the server will return NO_FILES_FOUND_ERROR indicating DM_MODIFIER_ID and DM_LAST_ACCESS_DATE were not set. If the resultant mask still contains a return value other than SUCCESSFUL, **NWSetNSEntryDOSInfo** will set the remaining bits and return SUCCESSFUL even though DM_MODIFIER_ID and DM_LAST_ACCESS_DATE were not set.

NCP Calls

0x2222 87 07 Modify File or Subdirectory DOS Information

NWSetNSInfo

Sets specific NS information for a directory entry specified by *volNum, nameSpace,* and *dirBase*

Local Servers: blocking

Remote Servers: blocking

Classification: 3.11, 3.12, 4.x

SMP Aware: No

Service: Name Space

Syntax

```
#include <nwfile.h>

int NWSetNSInfo (
   BYTE   volNum,
   BYTE   srcNameSpace,
   BYTE   dstNameSpace,
   LONG   dirBase,
   LONG   nsInfoMask,
   LONG   nsSpecificInfoLen,
   BYTE   *nsSpecificInfo);
```

Parameters

volNum

(IN) The volume number for which information is being set.

srcNameSpace

(IN) Name space that corresponds with the *dirBase* being passed. (The name space currently being worked with is the default.)

dstNameSpace

(IN) Name space to which information is being set.

dirBase

(IN) Directory base (or number) for the entry on which information is being set.

nsInfoMask

(IN) Bit map that indicates which types of information the user is setting in the data parameter.

nsSpecificInfoLen

(IN) Length of the data being set.

nsSpecificInfo

(IN) Data that is being set as indicated in the *nsInfoMask*.

Return Values

ESuccess or NetWare errors

Remarks

If the current name space is NFS, a value of 2 (for NFS) would be passed as *srcNameSpace*. If, however, the returned information should be in another format, for example OS/2, a value of 4 (for OS/2) would be passed as the *dstNameSpace*.

See DOS Name Space Bit Mask.

See Also

NWGetDirBaseFromPath, **NWGetNSInfo (NLM)**, **NWQueryNSInfoFormat**

NWWriteExtendedNSInfo

Writes the extended (huge) name space information for the specified name space

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Name Space

Syntax

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

NWCCODE N_API NWWriteExtendedNSInfo (
   NWCONN_HANDLE       conn,
   NW_IDX N_FAR       *idxStruct,
   NW_NS_INFO N_FAR   *NSInfo,
   pnuint8            data);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWWriteExtendedNSInfo
  (conn : NWCONN_HANDLE;
   Var idxStruct : NW_IDX;
   Var NSInfo : NW_NS_INFO;
   data : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

idxStruct

(IN) Points to NW_IDX returned by **NWNSGetMiscInfo**.

NSInfo

(IN) Points to NW_NS_INFO returned by **NWGetNSInfo**.

data

(IN) Points to a buffer containing the data to be written to the name space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	NO_MODIFY_PRIVILEGES

Remarks

dstNameSpace and *dstDirBase* in NW_IDX are used to determine what entry to use for the Write.

extendedBitMask in NW_NS_INFO is a read-only information field that should be preserved from **NWReadExtendedNSInfo**.

NCP Calls

0x2222 87 27 Set Huge NS Information

See Also

NWGetDirectoryBase, **NWGetNSInfo**, **NWNSGetMiscInfo**, **NWReadExtendedNSInfo**, **NWWriteExtendedNSInfo**

NWWriteNSInfo

Sets the specific name space information
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Name Space

Syntax

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

NWCCODE N_API NWWriteNSInfo (
   NWCONN_HANDLE       conn,
   NW_IDX N_FAR       *idxStruct,
   NW_NS_INFO N_FAR   *NSInfo,
   pnuint8            data);
```

Pascal Syntax

```
#include <nwnamspc.inc>

Function NWWriteNSInfo
  (conn : NWCONN_HANDLE;
   Var idxStruct : NW_IDX;
   Var NSInfo : NW_NS_INFO;
   data : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

idxStruct

(IN) Points to NW_IDX returned by **NWNSGetMiscInfo**.

NSInfo

(IN) Points to NW_NS_INFO returned by **NWGetNSInfo**.

data

(IN) Points to a 512-byte buffer containing the data to be written to the name space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

For name spaces other than DOS, **NWWriteNSInfo** is passed to the appropriate name space NLM on the server. For name space 0 (DOS), the server processes the request.

The actual format of the data is determined by the NLM on the server. Unless format for the data on the server is known, **NWWriteNSInfo** should not be used.

Avoid setting the first field of the name space information. This is generally the name and is intended to be read-only. To rename a file, call **NWSetLongName**.

NCP Calls

0x2222 87 25 Set NS Information

See Also

NWGetDirectoryBase, **NWGetNSInfo**, **NWNSGetMiscInfo**, **NWReadNSInfo**

SetCurrentNameSpace

Sets the name space that is to be used for parsing paths that are input to server functions

Local Servers: blocking

Remote Servers: N/A

Classification: 3.x, 4.x

SMP Aware: No

Service: Name Space

Syntax

```
#include <nwfile.h>

BYTE SetCurrentNameSpace (
   BYTE   newNameSpace);
```

Parameters

newNameSpace

(IN) Specifies the new name space.

Return Values

This function returns the old name space if successful. If the specified name space is not valid or is not supported on the current working volume (CWV) and current working directory (CWD), this function returns error code 255.

Remarks

The **SetCurrentNameSpace** function sets the name space to be used by the current thread group for parsing paths. This name space is used by this thread group for paths **input** to subsequent calls to functions from the NetWare API (until changed by another call to this function).

SetTargetNameSpace sets the name space for output from subsequent calls to functions from the NetWare API.

See Also

FEGetOriginatingNameSpace, **GetNameSpaceName**, **SetTargetNameSpace**

Example

SetCurrentNameSpace

```
#include <nwfile.h>
BYTE   oldNameSpace;
BYTE   newNameSpace;
oldNameSpace = SetCurrentNameSpace (newNameSpace);
```

SetTargetNameSpace

Sets the target name space that is to be returned by any following server functions

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.11, 3.12, 4.x

SMP Aware: No

Service: Name Space

Syntax

```
#include <nwfile.h>

BYTE SetTargetNameSpace (
   BYTE   newNameSpace);
```

Parameters

newNameSpace

(IN) Specifies the new name space that is to become the target name space.

Return Values

This function returns the old target name space.

Remarks

SetTargetNameSpace sets the target name space to be used by the current thread group. This name space is used by this thread group for paths **output** from all subsequent NetWare API functions.

SetCurrentNameSpace sets the name space for input to subsequent calls to functions from the NetWare API.

See Also

FEGetOriginatingNameSpace, **SetCurrentNameSpace**

Name Space: Structures

MODIFY_DOS_INFO

Defines the parameters for modifying an entry's DOS name space information

Service: Name Space

Defined In: nwnamspc.h

Structure

```
typedef struct
{
   nuint32   attributes;
   nuint16   createDate;
   nuint16   createTime;
   nuint32   creatorID;
   nuint16   modifyDate;
   nuint16   modifyTime;
   nuint32   modifierID;
   nuint16   archiveDate;
   nuint16   archiveTime;
   nuint32   archiverID;
   nuint16   lastAccessDate;
   nuint16   inheritanceGrantMask;
   nuint16   inheritanceRevokeMask;
   nuint32   maximumSpace;
} MODIFY_DOS_INFO;
```

Pascal Structure

```
Defined in nwnamspc.inc

 MODIFY_DOS_INFO = Record
    attributes : nuint32;
    createDate : nuint16;
    createTime : nuint16;
    creatorID : nuint32;
    modifyDate : nuint16;
    modifyTime : nuint16;
    modifierID : nuint32;
    archiveDate : nuint16;
    archiveTime : nuint16;
    archiverID : nuint32;
    lastAccessDate : nuint16;
    inheritanceGrantMask : nuint16;
    inheritanceRevokeMask : nuint16;
    maximumSpace : nuint32
 End;
```

Fields

attributes

Specifies the attributes to the value (see Attribute Values).

createDate

Specifies the creation date.

createTime

Specifies the creation time.

creatorID

Specifies the creator to the specified ID.

modifyDate

Specifies the date the entry was last modified.

modifyTime

Specifies the time the entry was last modified.

modifierID

Specifies the modifier to the specified ID.

archiveDate

Specifies the date the entry was last archived.

archiveTime

Specifies the time the entry was last archived.

archiverID

Specifies the archiver to the specified ID.

lastAccessDate

Specifies the date the entry was last accessed.

inheritanceGrantMask

Specifies the following TA constants:

C Value	Pascal Value	Value Name	Value Description
0x00	$00	TA_NONE	Specifies no Reads or Writes are allowed.
0x01	$01	TA_READ	Specifies file Reads are allowed.
0x02	$02	TA_WRITE	Specifies file Writes are allowed.
0x08	$08	TA_CREATE	Specifies files can be created.
0x10	$10	TA_DELETE	Specifies files can be deleted.
0x20	$20	TA_OWNERSHI	Specifies subdirectories can be

		P	created or deleted and trustee rights granted or revoked.
0x40	$40	TA_SEARCH	Specifies the directory can be searched.
0x80	$80	TA_MODIFY	Specifies file attributes can be modified.
0xFB	$FB	TA_ALL	Specifies the trustee has all the above rights to the directory.

inheritanceRevokeMask

Specifies the following TA constants:

C Value	Pascal Value	Value Name	Value Description
0x00	$00	TA_NONE	Specifies no Reads or Writes are allowed.
0x01	$01	TA_READ	Specifies file Reads are allowed.
0x02	$02	TA_WRITE	Specifies file Writes are allowed.
0x08	$08	TA_CREATE	Specifies files can be created.
0x10	$10	TA_DELETE	Specifies files can be deleted.
0x20	$20	TA_OWNERSHIP	Specifies subdirectories can be created or deleted and trustee rights granted or revoked.
0x40	$40	TA_SEARCH	Specifies the directory can be searched.
0x80	$80	TA_MODIFY	Specifies file attributes can be modified.
0xFB	$FB	TA_ALL	Specifies the trustee has all the above rights to the directory.

maximumSpace

NW_ENTRY_INFO

Holds standard name space information for an entry
Service: Name Space
Defined In: nwnamspc.h

Structure

```
typedef struct
{
   nuint32   spaceAlloc;
   nuint32   attributes;
   nuint16   flags;
   nuint32   dataStreamSize;
   nuint32   totalStreamSize;
   nuint16   numberOfStreams;
   nuint16   creationTime;
   nuint16   creationDate;
   nuint32   creatorID;
   nuint16   modifyTime;
   nuint16   modifyDate;
   nuint32   modifierID;
   nuint16   lastAccessDate;
   nuint16   archiveTime;
   nuint16   archiveDate;
   nuint32   archiverID;
   nuint16   inheritedRightsMask;
   nuint32   dirEntNum;
   nuint32   DosDirNum;
   nuint32   volNumber;
   nuint32   EADataSize;
   nuint32   EAKeyCount;
   nuint32   EAKeySize;
   nuint32   NSCreator;
   nuint8    nameLength;
   nstr8     entryName[256];
} NW_ENTRY_INFO;
```

Pascal Structure

```
Defined in nwnamspc.inc

  NW_ENTRY_INFO = Record
    spaceAlloc : nuint32;
    attributes : nuint32;
    flags : nuint16;
    dataStreamSize : nuint32;
    totalStreamSize : nuint32;
    numberOfStreams : nuint16;
```

```
    creationTime : nuint16;
    creationDate : nuint16;
    creatorID : nuint32;
    modifyTime : nuint16;
    modifyDate : nuint16;
    modifierID : nuint32;
    lastAccessDate : nuint16;
    archiveTime : nuint16;
    archiveDate : nuint16;
    archiverID : nuint32;
    inheritedRightsMask : nuint16;
    dirEntNum : nuint32;
    DosDirNum : nuint32;
    volNumber : nuint32;
    EADataSize : nuint32;
    EAKeyCount : nuint32;
    EAKeySize : nuint32;
    NSCreator : nuint32;
    nameLength : nuint8;
    entryName : Array[0..255] Of nstr8
End;
```

Fields

spaceAlloc

Specifies the space allocated to the datastream. IM_SPACE_ALLOC in *returnEntryInfo* mask.

attributes

Specifies the entry's attributes (see Attribute Values).

flags

Specifies data used internally.

dataStreamSize

Specifies the size of the datastream. IM_SIZE in *returnEntryInfo* mask.

totalStreamSize

Specifies the total size of streams associated with the entry. IM_TOTAL_SIZE in *returnEntryInfo* mask.

numberOfStreams

Specifies the number of streams associated with the entry.

creationTime

Specifies when the entry was created. IM_CREATION in *returnEntryInfo* mask.

creationDate

Specifies the date the entry was created (2 bytes). From the least significant byte to the most significant byte, 5 bits contains the day, 4 bits contains the month, and 7 bits contains the year.

creatorID

Specifies the object creating the entry.

modifyTime

Specifies the time the entry was last modified. IM_MODIFY in *returnEntryInfo* mask (2 bytes). From the least significant byte to the most significant byte, 5 bits contains the second, 6 bits contains the minute, and 5 bits contains the hour.

modifyDate

Specifies the date the entry was last modified (2 bytes). From the least significant byte to the most significant byte, 5 bits contains the day, 4 bits contains the month, and 7 bits contains the year.

modifierID

Specifies the ID of the object that last modified the entry.

lastAccessDate

Specifies the date the entry was last accessed (2 bytes). From the least significant byte to the most significant byte, 5 bits contains the day, 4 bits contains the month, and 7 bits contains the year.

archiveTime

Specifies the time the entry was last archived (2 bytes). IM_ARCHIVE in *returnEntryInfo* mask. From the least significant byte to the most significant byte, 5 bits contains the second, 6 bits contains the minute, and 5 bits contains the hour.

archiveDate

Specifies the date the entry was last archived (2 bytes). From the least significant byte to the most significant byte, 5 bits contains the day, 4 bits contains the month, and 7 bits contains the year.

archiverID

Specifies the ID of the object last archiving he entry.

inheritedRightsMask

Specifies the entry's inherited rights mask. IM_RIGHTS in *returnEntryInfo* mask. A mask of the following:

C Value	Pascal Value	Value Name	Value Description
0x00	$00	TA_NONE	Specifies no Reads or Writes are allowed.
0x01	$01	TA_READ	Specifies file Reads are allowed.
0x02	$02	TA_WRITE	Specifies file Writes are allowed.
0x08	$08	TA_CREATE	Specifies files can be created.

0x10	$10	TA_DELETE	Specifies files can be deleted.
0x20	$20	TA_OWNERSHIP	Specifies subdirectories can be created or deleted and trustee rights granted or revoked.
0x40	$40	TA_SEARCH	Specifies the directory can be searched.
0x80	$80	TA_MODIFY	Specifies file attributes can be modified.
0xFB	$FB	TA_ALL	Specifies the trustee has all the above rights to the directory.

dirEntNum

Specifies the directory entry number. IM_DIRECTORY in *returnEntryInfo* mask.

DosDirNum

Specifies the DOS directory entry number.

volNumber

Specifies the number of the volume that contains the entry.

EADataSize

Specifies the data size of the entry's extended attribute. IM_EA in *returnEntryInfo* mask.

EAKeyCount

Specifies the key count for the entry's extended attribute.

EAKeySize

Specifies the size of the entry's extended attribute key.

NSCreator

Specifies the name space the entry was originally created in. IM_OWNING_NAMESPACE in *returnEntryInfo* mask. Returns one of the following:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_OS2
4 NW_NS_LONG

nameLength

Specifies the length of the entry's name. IM_NAME in *returnEntryInfo* mask.

entryName

Specifies the entry's name.

NW_IDX

Receives the directory base for an entry in a specified name space
Service: Name Space
Defined In: nwnamspc.h

Structure

```
typedef struct
{
   nuint8    volNumber;
   nuint8    srcNameSpace;
   nuint32   srcDirBase;
   nuint8    dstNameSpace;
   nuint32   dstDirBase;
} NW_IDX;
```

Pascal Structure

```
Defined in nwnamspc.inc

  NW_IDX = Record
    volNumber : nuint8;
    srcNameSpace : nuint8;
    srcDirBase : nuint32;
    dstNameSpace : nuint8;
    dstDirBase : nuint32
  End;
```

Fields

volNumber

Specifies the volume number.

srcNameSpace

Specifies the name space of source:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_OS2
4 NW_NS_LONG

srcDirBase

Specifies the directroy base of source.

dstNameSpace

Specifies the name space changing to:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_OS2
4 NW_NS_LONG

dstDirBase

Specifies the directory base of the entry in the new name space.

NW_NS_INFO

Handles the information bit masks used to read name space-specific information

Service: Name Space

Defined In: nwnamspc.h

Structure

```
typedef struct
{
   nuint32   NSInfoBitMask;
   nuint32   fixedBitMask;
   nuint32   reservedBitMask;
   nuint32   extendedBitMask;
   nuint16   fixedBitsDefined;
   nuint16   reservedBitDefined;
   nuint16   extendedBitsDefined;
   nuint32   fieldsLenTable[32];
   nuint8    hugeStateInfo[16];
   nuint32   hugeDataLength;
} NW_NS_INFO;
```

Pascal Structure

```
Defined in nwnamspc.inc

 NW_NS_INFO = Record
    NSInfoBitMask : nuint32;
    fixedBitMask : nuint32;
    reservedBitMask : nuint32;
    extendedBitMask : nuint32;
    fixedBitsDefined : nuint16;
    reservedBitsDefined : nuint16;
    extendedBitsDefined : nuint16;
    fieldsLenTable : Array[0..31] Of nuint32;
    hugeStateInfo : Array[0..15] Of nuint8;
    hugeDataLength : nuint32
  End;
```

Fields

NSInfoBitMask

Specifies a bit mask with the following definitions:

C Value	Pascal Value	Value Name
0x0002L	$0002	DM_ATTRIBUTES

0x0004L	$0004	DM_CREATE_DATE
0x0008L	$0008	DM_CREATE_TIME
0x0010L	$0010	DM_CREATOR_ID
0x0020L	$0020	DM_ARCHIVE_DATE
0x0040L	$0040	DM_ARCHIVE_TIME
0x0080L	$0080	DM_ARCHIVER_ID
0x0100L	$0100	DM_MODIFY_DATE
0x0200L	$0200	DM_MODIFY_TIME
0x0400L	$0400	DM_MODIFIER_ID
0x0800L	$0800	DM_LAST_ACCESS_DATE
0x1000L	$1000	DM_INHERITED_RIGHTS_MASK
0x2000L	$2000	DM_MAXIMUM_SPACE

fixedBitMask

Specifies a bit mask representing fixed (sized) information.

reservedBitMask

Specifies a bit mask representing information stored as a length-preceded array. The first byte indicates the length.

extendedBitMask

Specifies a bit mask representing information stored as a length-preceded string with the first 2 bytes indicating the length.

fixedBitsDefined

Receives value indicating how many bits are defined within *fixedBitMask*.

reservedBitDefined

Receives value indicating how many bits are defined within *reservedBitMask*.

extendedBitsDefined

Receives value indicating how many bits are defined within *extendedBitMask*.

fieldsLenTable

Contains the length of the information relative to any of the three bit masks.Receives value indicating how many bits are defined within *reservedBitMask*.

hugeStateInfo

hugeDataLength

NW_NS_OPEN

is defined to be the same as the NW_NS_OPENCREATE structure
Service: Name Space
Defined In: nwnamspc.h

Remarks

The NW_NS_OPEN structure is defined to be the same as the NW_NS_OPENCREATE structure.

NW_NS_OPENCREATE

Defines the parameters for opening/creating a datastream in a specified name space

Service: Name Space

Defined In: nwnamspc.h

Structure

```
typdef struct
{
   nuint8    openCreateMode;
   nuint16   searchAttributes;
   nuint32   reserved;
   nuint32   createAttributes;
   nuint16   accessRights;
   nuint32   NetWareHandle;
   nuint8    openCreateAction;
} NW_NS_OPENCREATE
```

Pascal Structure

```
Defined in nwnamspc.inc

 NW_NS_OPENCREATE = Record
    openCreateMode : nuint8;
    searchAttributes : nuint16;
    reserved : nuint32;
    createAttributes : nuint32;
    accessRights : nuint16;
    NetWareHandle : nuint32;
    openCreateAction : nuint8
  End;
```

Fields

openCreateMode

Specifies whether to create, replace, or open an entry (directories can only be created). Open/Create modes use the OC_MODE_ constants listed below:

C Value	Pascal Value	Value Name
0x01	$01	OC_MODE_OPEN
0x02	$02	OC_MODE_TRUNCATE
0x02	$02	OC_MODE_REPLACE

0x08	$08	OC_MODE_CREATE

searchAttributes

Specifies the attributes to use in the search. Uses the SA_ constants listed below:

C Value	Pascal Value	Value Name
0x0000	$0000	SA_NORMAL
0x0002	$0002	SA_HIDDEN
0x0004	$0004	SA_SYSTEM
0x0010	$0010	SA_SUBDIR_ONLY
0x8000	$8000	SA_SUBDIR_FILES
0x8006	$8006	SA_ALL

reserved

createAttributes

Specifies the attributes to set in the DOS name space (see Attribute Values).

accessRights

Specifies the desired access rights.

NWHandle

Specifies a four-byte NetWare handle.

openCreateAction

Specifies the result of a successful open/create. Uses the OC_ACTION_ constants listed below:

C Value	Pascal Value	Value Name
0x01	$01	OC_ACTION_NONE
0x01	$01	OC_ACTION_OPEN
0x02	$02	OC_ACTION_CREATE
0x04	$04	OC_ACTION_TRUNCATE
0x04	$04	OC_ACTION_REPLACE

Remarks

To create a file, the *accessRights* field is used as an access rights mask and must be set to AR_READ and/or AR_WRITE. If neither are used, the NW_NS_OPENCREATE structure sets both. Use the AR constants listed

below:

C Value	Pascal Value	Value Name
0x0001	$0001	AR_READ
0x0002	$0002	AR_WRITE
0x0001	$0001	AR_READ_ONLY
0x0002	$0002	AR_WRITE_ONLY
0x0004	$0004	AR_DENY_READ
0x0008	$0008	AR_DENY_WRITE
0x0010	$0010	AR_COMPATABILITY
0x0040	$0040	AR_WRITE_THROUGH
0x0100	$0100	AR_OPEN_COMPRESSED

To create a directory, the*accessRights* field is used as an inherited rights mask and has the following bits:

Bit Number	Bit Definition
0	Read Existing File Bit
1	Write Existing File Bit
2	Old Open Existing File Bit
3	Create New Entry Bit
4	Delete Existing Bit
5	Change Access Control Bit
6	See Files Bit
7	Modify Entry Bit
8	Supervisor Privileges Bit
9-15	not set

NW_NS_PATH

Defines parameters for returning an entry's path with in a specified name space

Service: Name Space

Defined In: nwnamspc.h

Structure

```
typedef struct
{
   pnstr8    srcPath;
   pnstr8    dstPath;
   nuint16   dstPathSize;
} NW_NS_PATH;
```

Pascal Structure

```
Defined in nwnamspc.inc

  NW_NS_PATH = Record
    srcPath : pnstr8;
    dstPath : pnstr8;
    dstPathSize : nuint16
  End;
```

Fields

srcPath

Points to a valid path.

dstPath

Points to a buffer to receive the full name space path.

dstPathSize

Specifies the length of new path buffer. The new path buffer should be long enough to hold the longest path possible for *destNameSpace* plus 2 extra bytes for working space.

SEARCH_SEQUENCE

Defines information for managing a search operation across multiple requests

Service: Name Space

Defined In: nwnamspc.h

Structure

```
typedef struct
{
   nuint8    volNumber;
   nuint32   dirNumber;
   nuint32   searchDirNumber;
} SEARCH_SEQUENCE;
```

Pascal Structure

```
Defined in nwnamspc.inc

 SEARCH_SEQUENCE = Record
    volNumber : nuint8;
    dirNumber : nuint32;
    searchDirNumber : nuint32
  End;
```

Fields

volNumber

Specifies the volme number.

dirNumber

Specifies the directory entry number for the directory.

searchDirNumber

Specifies the directory number to search. Set to a 0xFFFFFFFF on the first call. After that, *searchDirNumber* is managed internally.

File Service Group

NetWare STREAMS

NetWare STREAMS: Guides

NetWare STREAMS: Concept Guide

NetWare STREAMS Introduction

NetWare STREAMS Functions

Additional Link

NetWare STREAMS: Functions

Parent Topic:

File Overview

NetWare STREAMS: Concepts

NetWare STREAMS Functions

Function	Purpose
getmsg	Retrieves the contents of a message.
ioctl	Performs a variety of control functions on a STREAM. See Ioctl Read\Write Client: Example.
poll	Monitors input and output on a set of file handles that reference open STREAMs.
putmsg	Creates a message.

Parent Topic:

NetWare STREAMS: Guides

NetWare STREAMS Introduction

NetWare® STREAMS is derived from UNIX System V™ Release 3.2.

A message format has been defined to simplify the design of service interfaces using NetWare STREAMS. The NetWare STREAMS functions enable user processes to create STREAMS messages and send them to neighboring kernel modules and drivers or receive the contents of such messages from kernel modules and drivers. These functions preserve message boundaries and provide separate buffers for the control and data parts of a message.

Parent Topic:

NetWare STREAMS: Guides

File Service Group

NetWare STREAMS: Functions

getmsg

Retrieves the contents of a message
Local Servers: blocking
Remote Servers: N/A
Classification: UNIX*
SMP Aware: No
Service: NetWare STREAMS

Syntax

```
#include <stropts.h>

int getmsg (
   int              handle,
   struct strbuf    *ctlptr,
   struct strbuf    *dataptr,
   int              *flags);
```

Parameters

handle

(IN) Specifies a file handle.

ctlptr

(OUT) Points to the structure containing the control part of the message.

dataptr

(OUT) Points to the structure containing the data part of the message.

flags

(IN) Specifies the priority of the message.

Return Values

Upon successful completion, a nonnegative value is returned. A value of 0 indicates that a full message was read successfully. A return value of MORECTL indicates that more control information is waiting for retrieval. A return value of MOREDATA indicates that more data is waiting for retrieval. A return value of MORECTLMOREDATA indicates that both types of information are waiting for retrieval. Subsequent **getmsg** calls retrieve the remainder of the message.

Remarks

The **getmsg** function retrieves the contents of a message located at the

stream-head read queue from a Stream file and places the contents into user-specified buffer(s). The message must contain either a data part, a control part, or both. The data and control parts of the message are placed into separate buffers. The semantics of each part is defined by the Stream module that generated the message.

The *handle* argument specifies a file handle referencing an open Stream. The *ctlptr* and *dataptr* arguments each point to a strbuf structure which contains the following members:

```
int    maxlen;   /* maximum buffer length */
int    len;      /* length of data */
char   *buf;     /* ptr to buffer */
```

where *buf* points to a buffer in which the data and control information is to be placed, and *maxlen* indicates the maximum number of bytes this buffer can hold. On return, *len* contains the number of bytes of data or control information actually received, or is 0 if there is a zero-length control or data part, or is -1 if no data or control information is present in the message.

By default, **getmsg** processes the first priority or nonpriority message available on the stream-head read queue. However, a user can choose to retrieve only priority messages by setting flags to RS_HIPRI. In this case, **getmsg** only processes the next message if it is a priority message.

If O_NDELAY has not been set, **getmsg** blocks until a message of the type (or types) specified by *flags* is available on the stream-head read queue. If O_NDELAY has been set and a message of the specified type is not present on the read queue, **getmsg** fails and sets *errno* to EAGAIN.

If a hangup occurs on the Stream from which messages are to be retrieved, **getmsg** continues to operate normally, as described above, until the stream-head read queue is empty. Thereafter, it returns a value of 0 in the *len* field of the *ctlptr* and *dataptr* arguments.

The **getmsg** function fails if one or more of the following is true:

AGAIN	The O_NDELAY flag is set and no messages are available.
EBADF	The *handle* argument is not a valid file handle open for reading.
EBADMSG	Queued message to be read is not valid for **getmsg**.
EFAULT	The *ctlptr*, *dataptr*, or *flags* argument points to a location outside the allocated address space.
EINTR	A signal was caught during the **getmsg** call.
EINVAL	An illegal value was specified in *flags*, or the Stream referenced by handle is linked under a multiplexer.
ENOSTR	A Stream is not associated with *handle*.

The **getmsg** function can also fail if a Stream error message had been received at the stream-head before the call to **getmsg**. The error returned is the value contained in the Stream error message.

See Also

putmsg, **read**, **write**

putmsg

Creates a message
Local Servers: blocking
Remote Servers: N/A
Classification: UNIX
SMP Aware: No
Service: NetWare STREAMS

Syntax

```
#include <stropts.h>

int putmsg (
   int              handle,
   struct strbuf    *ctlptr,
   struct strbuf    *dataptr,
   int              flags);
```

Parameters

handle

(IN) Specifies a file handle.

ctlptr

(OUT) Points to the structure containing the control part of the message.

dataptr

(OUT) Points to the structure containing the data part of the message.

flags

(IN) Specifies the priority of the message.

Return Values

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

Remarks

The **putmsg** function creates a message from user-specified buffers and sends the message to a Stream file. The message can contain a data part, a control part, or both. The data and control parts to be sent are distinguished by placement in separate buffers. The semantics of each part is defined by the Stream module that receives the message.

The *handle* argument specifies a file handle referencing an open Stream. The *ctlptr* and *dataptr* arguments each point to a strbuf structure which contains the following members:

```
int    maxlen;      /* maximum buffer length */
int    len;         /* length of data */
char   *buf;        /* ptr to buffer */
```

The *buf* field is a pointer to a buffer describing the control part (if any) to be included in the message. The *buf* field points to the buffer where the control information resides, and the *len* field indicates the number of bytes to be sent. The *maxlen* field is not used in **putmsg**. Similarly, *dataptr* specifies the data (if any) to be included in the message.

To send the data part of the message, *dataptr* must be nonNULL and the *len* field of *dataptr* must have a value of 0 or greater. To send the control part of a message, the corresponding values must be set for *ctlptr*. No data (control) part is sent if either *dataptr* (*ctlptr*) is NULL or the *len* field of *dataptr* (*ctlptr*) is set to -1.

If a control part is specified, and *flags* is set to RS_HIPRI, a priority message is sent. If *flags* is set to 0, a nonpriority message is sent. If no control part is specified, and *flags* is set to RS_HIPRI, **putmsg** fails and sets *errno* to EINVAL. If no control part and no data part are specified, and *flags* is set to 0, no message is sent, and 0 is returned.

For nonpriority messages, **putmsg** blocks if the Stream write queue is full due to internal flow control conditions. For priority messages, **putmsg** does not block on this condition. For nonpriority messages, **putmsg** does not block when the write queue is full and O_NDELAY is set. Instead, it fails and sets *errno* to EAGAIN.

The **putmsg** function also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the Stream, regardless of priority or whether O_NDELAY has been specified. No partial message is sent.

The **putmsg** function fails if one or more of the following is true:

EAGAIN	A nonpriority message was specified, the O_NDELAY flag is set, and the stream write queue is full due to internal flow control conditions.
EAGAIN	Buffers could not be allocated for the message that was to be created.
EBADF	The handle argument is not a valid file handle open for writing.
EFAULT	The *ctlptr*, *dataptr*, or *flags* parameter points to a location outside the allocated address space.
EINTR	A signal was caught during the **putmsg** call.
EINVAL	An undefined value was specified in *flags*, or *flags* is set to

	RS_HIPRI and no control part was supplied.
EINVAL	The Stream referenced by handle is linked below a multiplexer.
ENOSTR	A Stream is not associated with handle.
ENXIO	A hangup condition was generated downstream for the specified Stream.
ERANGE	The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost Stream module. This value is also returned if the control part of the message is larger than the maximum configured size of the control part of a message, or if the data part of the message is larger than the maximum configured size of the data part of a message.

The **putmsg** function can also fail if a Stream error message had been received at the stream-head before the call to **putmsg**. The error returned is the value contained in the Stream error message.

See Also

getmsg

Operating System I/O

Operating System I/O: Guides

Operating System I/O: Concept Guide

Operating System I/O Introduction

Operating System I/O File Paths

Operating System I/O Functions

Additional Link

Operating System I/O: Functions

Parent Topic:

File Overview

Operating System I/O: Concepts

File Permission Conversion

When a file is created during the operation of **creat**, **open**, or **sopen**, the *permission* parameter can be specified as S_IWRITE (writeable), S_IREAD (readable) or S_IWRITE | S_IREAD (writeable and readable). 0 also allows both writing and reading.

Files created with the S_IWRITE option can be written to, modified, and deleted by any object having such rights to the file. Files created with S_IREAD are created as read-only. S_IREAD is converted to directory attributes that prohibit writing, renaming, deletion, copying, or the ability to migrate or compress, as listed in File System Directory Entry Attributes.

Operating System I/O File Paths

When specifying a file to an Operating System I/O function, the file paths include the following conditions:

The file paths do not have drive letters.

File path drive letters are not used in NLM™ applications.

File paths can contain volume names.

Any volumes mounted on the server may be referenced by the NLM. Volume names are from 2 to 15 characters long.

The syntax for a file path that includes a volume name is as follows:

volume: directory\...\directory\filename

Each thread group in an NLM has its own current working directory (CWD). If a relative path is specified, it is assumed to be relative to the thread group's CWD. When a thread group is started, the initial value of the CWD is "SYS:" (root directory on the SYS volume). The only exception to this is when the (CLIB_OPT) parameters are specified when the NLM is loaded (see Using the LOAD Command in NLM Development: Concepts). These parameters change the CWD for the initial thread group.

NOTE: The maximum number of file handles that can be open at once for NetWare® 4.x is 1700.

NLM applications can open a given file more than once. The file handle and

NLM applications can open a given file more than once. The file handle and task information for subsequent opens depend on the circumstances of the open, as follows:

If a file is opened more than once by a particular thread, using the same connection and task, then the same file handle is returned. A count of the number of times a file is opened with a particular handle is associated with each handle. A handle remains usable as long as its open count is greater than zero.

If a file is opened more than once by different threads or by the same thread but with a different connection or task (than a previous connection or task with which the file was opened), then a different handle is returned. Closing one handle for a given file has no effect on any other handles to that file.

If a file is opened more than once with the same connection and task but with different threads, then the second (and subsequent) open is done with a newly allocated task number. The task number is automatically allocated on the current connection by **open** (or **sopen** or **creat**). The current task number is not affected.

I/O redirection on first-level files and file handles is supported **only** for NetWare 4.x (see **dup** and **dup2**). Redirection of second-level files is supported for all NetWare versions. Second-level files include those opened with **fopen**, **fdopen**, or **freopen** (see Stream I/O: Guides).

Text mode for first-level file handles is **not** supported. Only binary mode is supported. (In binary mode, data is transmitted unchanged. In text mode, carriage-return/line-feed pairs are translated to line feeds on input, and line feeds are translated to carriage-return/line-feed pairs on output.)

The following handles are predefined and always available:

STDIN (0)---Input from the current screen

STDOUT (1)---Output to the current screen

STDERR (2)---Output to the current screen

Parent Topic:

Operating System I/O: Guides

Operating System I/O Functions

Function	Purpose
chsize	Changes the file size.
close	Closes a file, stream, or BSD socket.
creat	Creates and opens a file or stream.

dup	Returns a file handle that refers to the same open file as handle. Supported only for the NetWare® 4.x OS.
dup2	Forces file handle handle2 to reference the same open file as handle Supported only for NetWare 4.x.
eof	Determines if the end of the file has been reached for a specified file.
fcntl	Provides control over open files.
filelength	Returns the number of bytes in an open file.
fstat	Obtains information about an open file.
isatty	Tests whether the specified handle refers to a screen or not.
lock	Locks a portion of a file.
lseek	Sets the current file position.
open	Opens a file, stream, or socket.
read	Reads data from a file, stream, or socket.
sopen	Opens a file, stream, or socket for shared access.
tell	Determines the current file position.
unlock	Unlocks a previously locked portion of a file.
write	Writes data to a file, stream, or socket.

Parent Topic:

Operating System I/O: Guides

Operating System I/O Introduction

Operating system I/O functions perform nonbuffered I/O operations. The functions in this section reference files using a file handle that is returned when a file is opened. The file handle is passed to the other functions. Files opened at the OS level (with the **open**, **sopen**, and **creat** functions), or opened at the stream level (see Stream I/O: Guides) and referenced with the **fileno** function are called **first-level** open files.

NOTE: As used in this chapter, streams are standard files and are not to be confused with NetWare® STREAMS.

For more information about NetWare STREAMS functions, see NetWare STREAMS: Guides.

NOTE: Operating System I/O provides functions for NLM development.

Parent Topic:

File Service Group

Operating System I/O: Guides

File Service Group

Operating System I/O: Functions

chsize

Changes the file size
Local Servers: blocking
Remote Servers: blocking
Classification: Other
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <unistd.h>

int chsize (
   int    handle,
   LONG   size);
```

Parameters

handle

(IN) Specifies a file handle.

size

(IN) Specifies the file size.

Return Values

chsize returns a value of 0 if successful. It returns a value of -1 if an error occurs.

If an error occurs, *errno* is set to:

4	EBADF	Bad file number.
6	EACCES	Permission denied.
12	ENOSPC	No space left on device.

If **chsize** does not complete successfully, *NetWareErrno* is set.

Remarks

The **chsize** function changes the size of the file associated with the file handle. It can truncate or extend the file, depending on the value of *size*

compared to the file's original size.

The mode in which the file was opened must allow writing.

If **chsize** extends the file, it appends NULL characters (\0). If it truncates the file, all data beyond the new end-of-file indicator is lost.

See Also

eof, **filelength**

close

Closes a file or stream
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <unistd.h>

int close (
   int   handle);
```

Parameters

handle

(IN) Specifies a file handle.

Return Values

When an error occurs while closing the file, a value of -1 is returned. Otherwise, a value of 0 is returned.

If an error occurs, *errno* is set to:

4	EBADF	Bad file number.

If the function does not complete successfully, *NetWareErrno* is set.

Remarks

This function also works on the DOS partition.

The *handle* value is the file handle returned by a successful execution of the **open**, **sopen**, or **creat** function. After a file is closed, the file handle is no longer valid and should not be reused.

UNIX STREAMS: If a Stream file is closed and the calling process had previously registered to receive a SIGPOLL signal for events associated with that file, the calling process is unregistered for events associated with the file.

The last **close** for a Stream causes the Stream associated with *handle* to be dismantled.

If O_NDELAY is not set and no signals have been posted for the Stream, the close function waits up to 15 seconds for each module and driver and for any output to drain before dismantling the Stream.

If the O_NDELAY flag is set or if there are any pending signals, the close function does not wait for output to drain and dismantles the Stream immediately.

See Also

creat, **dup**, **dup2**, **open**, **sopen**

creat

Creates and opens a file or stream
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <fcntl.h>

int creat (
   const char   *filename,
   int          permission);
```

Parameters

filename

(IN) Specifies the name of the file to be created.

permission

(IN) Specifies the access permissions for the file.

Return Values

When an error occurs while creating the file, a value of -1 is returned. Otherwise, an integer (not equal to -1), known as the file handle, is returned to be used with the other functions that operate on the file.

When an error occurs, *errno* can be set to:

1	ENONENT	No such file.
6	EACCES	Permission denied.
9	EINVAL	Invalid argument.

When an error occurs, *NetWareErrno* is set to:

152	(0x98)	ERR_INVALID_VOLUME	
15	(0x9C	ERR_INVALID_PATH	

6	)		
191	(0xBF)	ERR_INVALID_NAME_SPACE	On remote servers when using a non-DOS name space.

Remarks

This function also works on the DOS partition.

This function allows for as many open files as there is available memory. The *filename* parameter supplies the name of the file to be created. If the file exists (the current connection must have Write rights), it is truncated to contain no data and the preceding permission setting is unchanged.

If the file does not exist, it is created with access permission given by the *permission* parameter.

The access permission for the file is specified as a combination of bits (defined in the SYS\STAT.H header file):

S_IWRITE	The file is writeable.
S_IREAD	The file is readable.

The *permission* parameter can be specified as `S_IWRITE`, `S_IREAD` or `S_IWRITE | S_IREAD`. Specifiying 0 also makes a file both writeable and readable. File Permission Conversion provides further information about these options.

The current connection must have Create rights to create a new file or have Read/Write rights to write to a file that already exists.

On remote servers running NetWare 2.x, this function does not take partial paths.

See Also

dup, dup2, open, sopen

Example

creat

```
#include   <stddef.h>
#include   <fcntl.h>
```

```
#include   <errno.h>
#include   <string.h>

main()
{
   int   fh;
   if((fh = creat("name",0)) == -1)
      printf ("creat() error\n");
   else
      close (fh);
```

dup

Returns a file handle that refers to the same open file as handle (supported only for NetWare 4.0 and above)

Local Servers: nonblocking

Remote Servers: nonblocking

Classification: POSIX

SMP Aware: No

Service: Operating System I/O

Syntax

```
#include <unistd.h>

int dup (
   int   handle);
```

Parameters

handle

(IN) The file handle that is to be duplicated.

Return Values

When an error occurs, a value of -1 is returned. Otherwise, the return value is a nonnegative integer that is the file handle.

If an error occurs, *errno* can be set to:

4	EBADF	Bad file number
11	EMFILE	Too many open files

Remarks

The **dup** function duplicates a file handle by returning a file handle that refers to the same open file as *handle*. Since both handles reference the same file, either handle can be used for operations on the file.

NOTE: For an example of how to reverse the effect of redirecting *stdin*, see the example for **fdopen**.

See Also

close, creat, dup2, eof, fdopen, filelength, fileno, fstat, ftell, isatty, lseek, open, read, sopen, write

dup2

Forces the file handle to reference the same open file as the *handle* parameter (supported only for NetWare 4.0 and above)

Local Servers: nonblocking

Remote Servers: nonblocking

Classification: POSIX

SMP Aware: No

Service: Operating System I/O

Syntax

```
#include <unistd.h>

int dup2 (
   int   handle,
   int   handle2);
```

Parameters

handle

(IN) Specifies the file handle to be duplicated.

handle2

(IN) Specifies the file handle to be forced to reference the same file as the *handle* parameter.

Return Values

When an error occurs, a value of -1 is returned. Otherwise, the return value is a nonnegative integer that is the file handle.

If an error occurs, *errno* can be set to:

4	EBADF	Bad file number
11	EMFILE	Too many open files

Remarks

The *handle* parameter must be a handle to a file that is already open. If *handle2* references a file that is already open, that file is closed before *handle2* is forced to reference the file for *handle*.

See Also

close, creat, dup, eof, filelength, fileno, fstat, ftell, isatty, lseek, open, read, sopen, write

eof

Determines if the end of the file has been reached for a specified file
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <unistd.h>

int eof (
   int   handle);
```

Parameters

handle

(IN) Specifies a file handle.

Return Values

eof returns a value of 1 if the current file position is at the end of the file. If the current file position is not at the end, a value of 0 is returned. If an error is detected, a value of -1 is returned.

If an error occurs, *errno* is set to:

4	EBADF	Bad file number.

If **eof** does not complete successfully, *NetWareErrno* is set.

Remarks

The **eof** function determines if the end of the file has been reached for the file whose file handle is given by *handle*. Because the current file position is set following an input operation, **eof** can be called to detect the end of the file before an input operation beyond the end of the file is attempted.

See Also

dup, **dup2**, **read**

fcntl

Controls file handles
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <fcntl.h>

int fcntl (
   int   handle,
   int   cmd,
   int   arg);
```

Parameters

handle

(IN) Specifies a file handle to be operated on by *cmd*.

cmd

(IN) Specifies one of two commands to act on the specified file handle.

arg

(IN/OUT) Receives the handle status flags.

Return Values

Upon successful completion of the F_GETFL command, **fcntl** returns the current value of the requested flag. Otherwise, a value of -1 is returned and *errno* indicates the error.

4	EBADF	The specified file handle is not a valid one.
9	EINVAL	Either *cmd* or *val* is not supported.
35	EWOULDBLOCK	Either no data is available to a read call or a write operaton is in a blocking mode.

Remarks

The **fcntl** function provides for file control over file handles. The *handle* parameter is a file handle to be operated on by *cmd*. The *cmd* parameter

includes either the F_GETFL or the F_SETFL commands described below:

F_GETFL	Get handle status flags.
F_SETFL	Set handle status flags.

Flags are passed in the *arg* parameter. The FNDELAY flag is defined for the F_GETFL and F_SETFL commands. It establishes a nonblocking I/O mode; if no data is available to a read call or if a write operation is in a blocking mode, the call returns a value of -1 with the error EWOULDBLOCK.

See Also

ioctl

filelength

Returns the number of bytes in an open file
Local Servers: blocking
Remote Servers: blocking
Classification: Other
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <fcntl.h>

LONG filelength (
   int   handle);
```

Parameters

handle
(IN) Specifies a file handle.

Return Values

filelength returns a value of -1 if an error occurs.

If an error occurs, *errno* can be set to:

4	EBADF	Bad file number.

If **filelength** does not complete successfully, *NetWareErrno* is set.

Remarks

The **filelength** function returns the number of bytes in the opened file indicated by the file handle.

See Also

dup, **dup2**, **eof**, **lseek**, **tell**

Example

filelength

```
#include <fcntl.h>

LONG   length;
int    handle;
length = filelength (handle);
```

fstat

Obtains information about an open file
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <sys\stat.h>

int fstat (
   int           handle,
   struct stat   *statblk);
```

Parameters

handle

(IN) Specifies a file handle.

statblk

(OUT) Receives a pointer to the address of the structure stat.

Return Values

fstat returns a value of 0 when the information is obtained successfully. Otherwise, a value of -1 is returned.

If an error occurs, *errno* is set to:

4	EBADF	Bad file number.

If **fstat** does not complete successfully, *NetWareErrno* is set.

Remarks

The **fstat** function obtains information about an open file whose file handle is *handle*. This information is placed in the structure located at the address indicated by the *statblk* parameter.

The SYS\STAT.H header file contains definitions for stat (Structure) and describes the contents of the fields within that structure.

See Also

dup, dup2, open, stat (Function)

Example

fstat

```
#include <stdio.h>
#include <nwtypes.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>

main()
{
   int           handle;
   struct stat   buf;
   handle = open ("test.dat",O_RDONLY | O_BINARY,0);
   if(handle == -1)
   {
      printf ("could not open file");
      exit (0);
   }
   if(fstat (handle,&buf) == -1)
      printf ("fstat error\r\n");
   close(handle);
   printf ("st_dev = %x\r\n",buf.st_dev);
   printf ("st_ino = %x\r\n",buf.st_ino);
   printf ("st_mode = %x\r\n",buf.st_mode);
   printf ("st_nlink = %x\r\n",buf.st_nlink);
   printf ("st_uid = %x\r\n",buf.st_uid);
   printf ("st_gid = %x\r\n",buf.st_gid);
   printf ("st_rdev = %x\r\n",buf.st_rdev);
   printf ("st_size = %x\r\n",buf.st_size);
   printf ("st_atime = %x\r\n",buf.st_atime);
   printf ("st_mtime = %x\r\n",buf.st_mtime);
   printf ("st_ctime = %x\r\n",buf.st_ctime);
   printf ("st_btime = %x\r\n",buf.st_btime);
   printf ("st_attr = %x\r\n",buf.st_attr);
   printf ("st_archivedID = %x\r\n",buf.st_archivedID);
   printf ("st_updatedID = %x\r\n",buf.st_updatedID);
   printf ("st_inheritedRightsMask = %x\r\n",buf.st_inheritedRightsMask
   printf ("st_originatingNameSpace = %c\r\n",buf.st_originatingNameSpa
   printf ("st_name = %s\r\n",buf.st_name);
/*----------------- new fields starting in v. 4.11 ----------------*/
   printf ("st_name2 = %s\r\n",buf.st_name2);
   printf ("st_blksize = %x\r\n",buf.st_blksize);
   printf ("st_blocks = %x\r\n",buf.st_blocks);
   printf ("st_flags = %x\r\n",buf.st_flags);
```

```
    printf ("st_flags = %x\r\n",buf.st_flags);
}
```

ioctl

Performs a variety of control functions on a STREAMS, BSD Socket, and pipe file descriptors
Local Servers: blocking
Remote Servers: N/A
Classification: UNIX nonstandard
NetWare Server: 3.12, 4.x, 5.0
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <sys/ioctl.h>
#include <stropts.h>

int ioctl (
   int    filedes,
   int    command,
   void  *arg);
```

Parameters

filedes

(IN) Specifies a descriptor returned from the **open**, **pipe**, **sopen**, **socket**, etc. functions.

command

(IN) Specifies the control function to be performed.

arg

(IN/OUT) Specifies an additional argument to be used or points to an argument returned by **ioctl** (depending on the control function performed).

Return Values

Upon successful completion, the value returned depends upon the control function (command argument) but must be a nonnegative integer. Otherwise, *errno* indicates the occurring error.

Remarks

For Stream files, **ioctl** performs the following control operations:

FIOGETNBI	Description pending.

O		
FIONBIO	Description pending.	
FIOREAD	Description pending.	
I_FDINSERT	Creates a message from user-specified buffers, adds information about another Stream and sends the message downstream. The message contains a control part and an optional data part. On failure, *errno* is set.	
I_FIND	Compares the names of all modules currently present in the Stream to the name pointed to by the *arg* parameter, and returns a value of 1 if the named module is present in the Stream. It returns a value of 0 if the named module is not present. On failure, *errno* is set.	
I_FLUSH	Flushes all input and/or output queues, depending on the value of the *arg* parameter :	
	FLSHR	Flush read queues.
	FLUSHW	Flush write queues.
	FLUSHRW	Flush read and write queues.
I_GETSIG	Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by the *arg* parameter, where the events are those specified in the description of I_SETSIG. On failure, *errno* is set.	
I_GRDOPT	Returns the current read mode setting in an int pointed to by the *arg* parameter. On failure, *errno* is set.	
I_LINK	Connects two Streams, where the *filedes* parameter contains the file descriptor of the Stream connected to the multiplexing driver, and the *arg* parameter is the file descriptor of the Stream connected to another driver. The Stream designated by the *arg* parameter is connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgment message to the stream-head regarding the linking operation. It returns a multiplexer ID number (an identifier used to disconnect the multiplexer) on success, and a value of -1 on failure. On failure, *errno* is set.	
I_LOOK	Retrieves the name of the module just below the stream-head of the Stream pointed to by the *filedes* parameter, and places it in a NULL-terminated character string pointed to by the *arg* parameter. On failure, *errno* is set.	
I_NREAD	Counts the number of data bytes in data blocks in the first message on the stream-head read queue, and	

	places this value in the location pointed to by the *arg* parameter. The return value for the command is the number of messages on the stream-head read queue. On failure, *errno* is set. For a pipe file, counts the number of data bytes left to read on the descriptor before the **read** function blocks.
I_NWRITE	Counts the number of data bytes that may be written on a pipe descriptor before the **write** function blocks. Similar to I_NREAD. For example, err=ioctl(pipeFD, I_NWRITE, &pending);.
I_PEEK	Allows a user to retrieve the information in the first message on the stream-head read queue without taking the message off the queue. The *arg* parameter points to a strpeek structure. I_PEEK returns a value of 1 if a message was retrieved, and returns a value of 0 if no message was found on the stream-head read queue.
I_POP	Removes the module just below the stream-head of the Stream pointed to by the *filedes* parameter. The *arg* parameter should be 0 in an I_POP request. On failure, *errno* is set.
I_PUSH	Pushes the module whose name is pointed to by the *arg* parameter onto the top of the current Stream, just below the stream-head. It then calls the open routine of the newly pushed module. On failure, *errno* is set.
I_RECVFD	Retrieves the file descriptor associated with the message sent by an I_SENDFD **ioctl** over a stream pipe. The *arg* parameter points to a data buffer large enough to hold the strrecvfd structure. If the message at the stream-head is a message sent by an I_SEND descriptor, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the strrecvfd structure. The structure is copied into the user data buffer pointed to by the *arg* parameter. On failure, *errno* is set.
I_SENDFD	Requests the Stream associated with the file descriptor to send a message, containing a file pointer, to the stream-head at the other end of a stream pipe. The *arg* parameter must point to a file descriptor. I_SENDFD converts the *arg* parameter into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user ID and group ID associated with the sending process are also inserted. This message is placed directly on the read queue for the stream-head at the other end of the stream pipe to which it is connected. On failure,*errno* is set.
I_SETBUF	Exchanges the buffer currently underlying the pipe for one of a different size. The pipe becomes empty and

	cleared with respect to data read or written. For example, err=ioctl(pipeFD, I_SETBUF, 8192);.	
I_SETSIG	Informs the stream-head that the user wants the kernel to issue the SIGPOLL signal when a particular event has occurred on the Stream associated with the *filedes* parameter. I_SETSIG supports an asynchronous processing capability in STREAMS. The value of the *arg* parameter is a bitmask that specifies the events for which the user should be signaled. It is the bitwise OR of any combination of the following constants:	
	S_INPUT	Nonpriority message has arrived.
	S_HIPRI	Priority message is present.
	S_OUTPUT	The write queue is no longer full. There is room for sending (or writing) data downstream.
	S_MSG	Stream signal message containing the SIGPOLL signal has reached the front of the stream-head read queue. On failure, *errno* is set.
I_SRDOPT	Sets the read mode using the *arg* parameter. Legal values for the *arg* parameter are:	
	RNORM	Byte-stream mode (the default)
	RMSGD	Message-discard mode
	RMSGN	Message-nondiscard mode
I_STR	Constructs an internal Stream **ioctl** message from the data pointed to by the *arg* parameter and sends that message downstream. I_STR blocks until the system responds with either a positive or negative acknowledgment message or until the request times out after some period of time. If the request fails, *errno* is set.	
I_UNLINK	Disconnects the two Streams specified by the *filedes* and *arg* parameters. The *filedes* parameter is the file descriptor of the Stream connected to the multiplexing driver. The descriptor must correspond to the Stream on which the **ioctl** I_LINK command was issued to link the Stream below the multiplexing driver. The *arg* parameter is the multiplexer ID number that was returned by the I_LINK call. If the *arg* parameter is -1, all Streams which were linked to the *filedes* parameter are disconnected. As in I_LINK, it requires the multiplexing driver to acknowledge the unlink. On failure, *errno* is set.	

IP_INBOUND_IF	Description pending.
IP_OUTBOUND_IF	Description pending.
SIOCATMARK	Description pending.
SIOCDGRAMSIZE	Description pending.

See Ioctl Read\Write Client: Example.

See Also

pipe, poll, putmsg, open, read, socket, write

isatty

Tests whether the specified handle refers to a screen
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <unistd.h>

int isatty (
   int   handle);
```

Parameters

handle
(IN) Specifies a file handle.

Return Values

isattyn returns a value of 0 if the device or file is not a character device; otherwise, a nonzero is returned.

If an error occurs,*errno* can be set to:

4	EBADF	Bad file number.

Remarks

The **isatty** function tests if the opened file or device referenced by the file handle is a character device (namely, the console).

See Also

dup, **dup2**, **open**

lock

Locks a portion of a file
Local Servers: blocking
Remote Servers: blocking
Classification: Other
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <nwfile.h>

int lock (
   int    handle,
   LONG   offset,
   LONG   length);
```

Parameters

handle

(IN) Specifies a file handle.

offset

(IN) Specifies the starting byte that is to be locked.

length

(IN) Specifies the amount of data (in bytes) to be locked.

Return Values

Returns a value of 0 if successful, and a value of -1 when an error occurs.

If an error occurs, *errno* is set to:

4	EBADF	Bad file number.

Remarks

lock locks the amount of data specified by the *length* parameter in the file specified by the *handle* parameter, starting at the byte specified by the *offset* parameter in the file.

lock prevents other open handles from reading or writing into the locked

region until an unlock has been done for this locked region of the file. All locked regions of a file must be unlocked before a file is closed.

See Also

open, **sopen**, **unlock**

lseek

Sets the current file position
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <unistd.h>

LONG lseek (
   int    handle,
   LONG   offset,
   int    origin);
```

Parameters

handle

(IN) Specifies a file handle.

offset

(IN) Specifies the relative offset from a file position.

origin

(IN) Specifies the seek starting point.

Return Values

lseek returns a value of -1 when an error occurs. Otherwise, the new current file position is returned in a system-dependent manner. A value of 0 indicates the start of a file.

If an error occurs,*errno* is set to:

4	EBADF	Bad file number.

If **lseek** does not complete successfully,*NetWareErrno* is set.

Remarks

The file is referenced using the specified file handle.

The value of the *offset* parameter is used as a relative offset from a file position determined by the value of the *origin* parameter. An absolute offset can be from 0 to 2^{32}. It must be unsigned long, which cannot be negative.

The new file position is determined in a manner dependent upon the value of the *origin* parameter, which can have one of three possible values (defined in the STDIO.H header file):

SEEK_SET	The new file position is computed relative to the start of the file.
SEEK_CUR	The new file position is computed relative to the current file position.
SEEK_END	The new file position is computed relative to the end of the file.

The files's position can be set to a position outside of the bounds of the file.

See Also

close, **creat**, **dup**, **dup2**, **eof**, **filelength**, **fileno**, **fstat**, **isatty**, **open**, **read**, **sopen**, **tell**, **write**

Example

lseek

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

main()
{
   int   fh;
   if((fh = open ("test.dat",O_RDWR | O_CREAT | O_TRUNC,0)) < 0)
   {
      printf ("could not open file\r\n");
      exit(0);
   }
   write (fh,"1234567890",10);
   if(lseek (fh,1,SEEK_SET) < 0)
   {
      printf ("error on seek 1\r\n");
```

```
      goto end;
   }
   write (fh,"a",1);
   if(lseek(fh,-2,SEEK_END) < 0)
   {
      printf ("error on seek 2\r\n");
      goto end;
   }
   write (fh,"b",1);
   if(lseek (fh,-5,SEEK_CUR) < 0)
   {
      printf ("error on seek 3\r\n");
      goto end;
   }
   write (fh,"c",1);
   end:
   close (fh);
   getch ();
}
```

open

Opens a file or stream
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <fcntl.h>

int open (
   const char   *path,
   int          oflag,
   ...);
```

Parameters

path

(IN) Specifies the name of the file to open.

oflag

(IN) Specifies the access mode.

Return Values

When an error occurs while opening the file, a value of -1 is returned. Otherwise, an integer (not equal to -1), known as the file handle, is returned to be used with the other functions that operate on the file.

If an error occurs, *errno* can be set to:

1	ENONENT	No such file.
6	EACCES	Permission denied.
9	EINVAL	Invalid argument.

When an error occurs, *NetWareErrno* is set to:

108	(0x6C)	ERR_BAD_ACCESS

152	(0x98)	ERR_INVALID_VOLUME
156	(0x9C)	ERR_INVALID_PATH

Remarks

This function also works on the DOS partition.

This function allows for as many open files as there is available memory. The *path* parameter supplies the name of the file to be opened. The file is accessed according to the access mode specified by the *oflag* parameter.

The access mode is established as a combination of the bits defined in the FCNTL.H header file. The following bits can be set:

O_RDONLY	Permits the file to be only read.
O_WRONLY	Permits the file to be only written.
O_RDWR	Permits the file to be both read and written.
O_APPEND	Causes each record that is written to be written at the end of the file.
O_CREAT	Has no effect when the file indicated by the filename parameter already exists; otherwise, the file is created.
O_TRUNC	Causes the file to be truncated to contain no data when the file exists; has no effect when the file does not exist. O_TRUNC must be ORed with write access to truncate a file: O_TRUNC | O_RDWR O_TRUNC | O_WRONLY
O_BINARY	Causes the data to be transmitted unchanged. (Text mode for first-level handles is not supported. In text mode, carriage- return/line-feed pairs are translated to line feeds on input, and line feeds are translated to carriage-return/line-feed pairs on output.)

O_CREAT must be specified when the file does not exist and it is to be written.

An optional third parameter, int *permission*, is used when the file is to be created (O_CREAT is specified) to set file permissions. File permissions are set according to the value contained in the *permission* parameter. The access permissions for the file is specified as a combination of bits (defined in the SYS\STAT.H header file).

S_IWRITE	The file is writeable.
S_IREAD	The file is readable.

The *permission* parameter can be specified as `S_IWRITE`, `S_IREAD` or `S_IWRITE | S_IREAD`. Specifiying 0 also makes a file both writeable and readable. File Permission Conversion provides further information about these options.

On remote servers running NetWare 2.x, this function does not take partial paths.

UNIX Streams: When opening a Stream file, access must be constructed from O_NDELAY and either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to Stream devices and have no effect on them. The value of O_NDELAY affects the operation of Stream drivers and certain function calls (**read**, **getmsg**, **putmsg**, and **write**). For drivers, the implementation of O_NDELAY is device-specific. Each Stream device driver can treat this option differently.

SetCurrentNameSpace sets the name space which is used for parsing the path input to this function.

See Also

close, **creat**, **dup**, **dup2**, **eof**, **filelength**, **fileno**, **fstat**, **isatty**, **lseek**, **read**, **sopen**, **tell**, **write**

Example

open

```
#include <stddef.h>
#include <fcntl.h>
#include <errno.h>

main()
{
   int fh,size;
   char buffer[] = {"a text record to be written\n" };
   fh = open ("test.dat",O_WRONLY | O_CREAT | O_TRUNC,0);
   printf ("handle: %d\n\r",fh);
   if(fh == EFAILURE)
   {
      printf ("could not open file\r\n");
      goto end;
   }
```

```
   printf ("%d\n\r",tell(fh));
   size = write (fh,buffer,sizeof(buffer));
   if(size < 29)
   {
      printf ("could not write to file\r\n");
      goto end;
   }
   printf ("%d\r\n",tell(fh));
   close (fh);
   end:
   printf ("%d\r\n",errno);
   getch ();
}
```

pipe

Local Servers: blocking
Remote Servers: N/A
Classification: UNIX nonstandard
NetWare Server: 3.12, 4.x, 5.0
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <sys/ioctl.h>
#include <stropts.h>

int pipe (
   ,
   ,
   );
```

Parameters

Return Values

Upon successful completion, the value returned depends upon the control function (command argument) but must be a nonnegative integer. Otherwise, *errno* indicates the occurring error.

Remarks

See Also

ioctl, **poll**, **putmsg**, **open**, **read**, **socket**, **write**

read

Reads data from a file or stream
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <unistd.h>

LONG read (
   int    handle,
   char   *buffer,
   LONG   len);
```

Parameters

handle

(IN) Specifies a file handle.

buffer

(OUT) Points to a buffer to receive the data.

len

(IN) Specifies the number of bytes to read.

Return Values

read returns the number of bytes of data transmitted from the file to the buffer. Normally, this is the number given by the *len* parameter. When the end-of- file is encountered before the read completes, the return value is less than the number of bytes requested.

A value of -1 is returned when an input/output error is detected. If an error occurs, *errno* can be set to:

4	EBADF	Bad file number.

If **read** does not complete successfully, *NetWareErrno* is set.

Remarks

This function also works on the DOS partition.

The **read** function returns the number of bytes of data transmitted from the file to the buffer.

The *handle* value is returned by the **open**, **sopen**, **creat**, or **fileno** function. The access mode must have included either O_RDONLY or O_RDWR when the open or **sopen** function was invoked. The data is read starting at the current file position for the file in question. This file position can be determined with the tell function and can be set with the **lseek** function.

UNIX STREAMS

A read from a Stream file can operate in three different modes: byte-stream mode, message-nondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the I_SRDOPT **ioctl** request, and can be tested with the I_GRDOPT **ioctl**. In byte-stream mode, the read function retrieves data from the Stream until it has received *nbyte* bytes, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In Stream message-nondiscard mode, the **read** function retrieves data until it has read *nbytes* bytes or until it reaches a message boundary. If the **read** function does not retrieve all the data in the message, the remaining data are placed on the Stream, and can be retrieved by the next **read** or **getmsg** call. Message-discard mode also retrieves data until it has retrieved *nbyte* bytes or it reaches a message boundary. However, unread data remaining in a message after the read function returns are discarded and are not available for a subsequent **read** or **getmsg**.

When reading from a Stream file, handling of zero-byte messages is determined by the current read mode setting. In byte-stream mode, the read function accepts data until it has read *nbyte* bytes, or until there is no more data to read or until a zero-byte message block is encountered. The read function returns the number of bytes read and places the zero-byte message back on the Stream to be retrieved by the next **read** or **getmsg**. In the two other modes, a zero-message returns a value of 0 and the message is removed from the Stream. When a zero-byte message is read as the first message on a Stream, a value of 0 is returned regardless of the read mode.

A read from a Stream file can only process data messages. It cannot process any type of protocol message and fails if a protocol message is encountered at the stream head.

A read from a Stream file also fails if an error message is received at the stream head. In this case, *errno* is set to the value returned in the error message. If a hangup occurs on the Stream being read, the read function continues to operate normally until the stream head read queue is empty. Thereafter, it returns 0.

See Also

close, creat, dup, dup2, eof, filelength, fileno, fstat, isatty, lseek, open, sopen, tell, write

Example

read

```
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <nwconio.h>

#define BUFFER_SIZE 512

main()
{
   int     handle, rc, i, ch = 0;
   char    buffer[BUFFER_SIZE];
   handle = open ("test.dat", O_RDONLY, 0 );
   if (handle < 0)
   {
      printf ("\r%s\r\n\n",strerror(errno));
      exit (0);
   }
   while (1)
   {
      if ((rc = read (handle, buffer, BUFFER_SIZE)) <= 0)
      {
         lseek (handle,0,SEEK_SET);
         rc = 0;
      }
      for (i = 0; i < rc; i++)
      {
         putchar (buffer[i] );
         if (kbhit ())
         {
         if ((ch = getch()) == 0x03)
         {
            printf ("^C" );
            close (handle );
            exit (0);
         }
         else if(ch == 'l')
         {
            printf ("\r\n\n***** lock %s\r\n\n",
            lock (handle,1,10) ? "failed" : "succeeded");
```

```
                getch ();
             }
             else if(ch == 'u')
             {
                printf ("\r\n\n***** unlock %s\r\n\n",
                unlock(handle,1,10) ? "failed" : "succeeded");
                getch ();
             }
             else getch (); /* Pause*/
             }
          }
       }
    }
```

setmode

Sets, at the operating system level, the translation mode to the specified value

Local Servers: nonblocking

Remote Servers: nonblocking

Classification: Other

SMP Aware: No

Service: Operating System I/O

Syntax

```
#include <fcntl.h>

int setmode (
   int   handle,
   int   mode);
```

Parameters

handle

(IN) Specifies a file handle.

mode

(IN) Specifies the translation mode.

Return Values

If successful, the **setmode** function returns the previous mode that was set for the file. Otherwise, a value of -1 is returned. When an error has occurred, the global variable *errno* contains a value indicating the type of error that has been detected.

Remarks

The **setmode** function sets the translation mode to be the value of mode for the file whose file handle is given by handle. The *mode* parameter can contain the following value:

O_BINARY---Data is read or written unchanged.

See Also

close, **creat**, **eof**, **filelength**, **fileno**, **fstat**, **isatty**, **lseek**, **open**, **read**, **sopen**, **tell**, **write**

sopen

Opens a file or stream for shared access
Local Servers: blocking
Remote Servers: blocking
Classification: Other
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <fcntl.h>

int sopen (
   const char   *filename,
   int          access,
   int          share,
   int          permission);
```

Parameters

filename

(IN) Specifies the name of the file to open.

access

(IN) Specifies the access mode.

share

(IN) Specifies the sharing mode of the file.

permission

(IN) Specifies the file permissions.

Return Values

When an error occurs while opening the file, a value of -1 is returned. Otherwise, an integer (not equal to -1), known as the file handle, is returned to be used with the other functions that operate on the file.

When an error occurs, *errno* is set to:

1	ENOENT	No such file.
6	EACCESS	Permission denied.
9	EINVAL	Invalid argument.

When an error occurs, *NetWareErrno* is set to:

107	(0x6B)	ERR_BAD_SHFLAG
108	(0x6C)	ERR_BAD_ACCESS
152	(0x98)	ERR_INVALID_VOLUME
156	(0x9C)	ERR_INVALID_PATH

NOTE: A problem was introduced in the 4.01d version of clib.nlm. It manifests itself when **sopen** is called multiple times by the same thread.

In clib.nlm versions prior to 4.01d, a valid file handle will be returned from **sopen** when a file is opened multiple times by the same thread. In the 4.01d version of clib.nlm (and later versions) an error occurs and returns -1 while errno is set to EINUSE.

Beginning with version 4.10d (notice this is 4.1 and not 4.01) PTR 2/1/95, you can pass the NWSH_PRE_401D_COMPAT bit (found in share.h) in as part of the share flags passed to **sopen** to get functionality of clib versions prior to 4.01d (if you have experienced problems with multiple threads opening the same file multiple times).

As an alternative to the above work around, you can pass in a CLIB_OPT switch on the command line of the .nlm you are using. CLIB_OPT/U86414 will provide the same functionality as passing in the share flags with the additional bits ORed in. This is only available on clib.nlm version 4.10d PTF.

Remarks

This function also works on the DOS partition.

The name of the file to be opened is given by the *filename* parameter. The file is accessed according to the access mode specified by the*access* parameter.

The access mode is established as a combination of the bits defined in the FCNTL.H header file.

The following bits can be set:

O_RDONLY	Permits the file to be only read.

O_WRONLY	Permits the file to be only written.
O_RDWR	Permits the file to be both read and written.
O_APPEND	Causes each record that is written to be written at the end of the file.
O_CREAT	Has no effect when the file indicated by the filename parameter already exists; otherwise, the file is created.
O_TRUNC	Causes the file to be truncated to contain no data when the file exists; has no effect when the file does not exist. O_TRUNC must be ORed with write access to truncate a file:
	O_TRUNC | O_RDWR
	O_TRUNC | O_WRONY
O_BINARY	Causes the data to be transmitted unchanged. (Text mode for first-level handles is not supported. In text mode, carriage- return/line-feed pairs are translated to line feeds on input, and line feeds are translated to carriage-return/line-feed pairs on output.)

O_CREAT must be specified when the file does not exist and it is to be written.

When the file is to be created (O_CREAT is specified), the file permissions are set according to the value contained in the *permission* parameter.

The access permissions for the file is specified as a combination of bits (defined in the SYS\STAT.H header file).

S_IWRITE	The file is writeable.
S_IREAD	The file is readable.

The *permission* parameter can be specified as `S_IWRITE`, `S_IREAD` or `S_IWRITE|S_IREAD`. Specifiying 0 also makes a file both writeable and readable. File Permission Conversion provides further information about these options.

The shared access for the file is established by the combination of bits set in the *share* parameter where the following values are defined in NWSHARE.H.

SH_COMP	Sets compatibility mode.

AT	
SH_DENY RW	Prevents read or write access to the file.
SH_DENY WR	Prevents write access of the file.
SH_DENY RD	Prevents read access to the file.
SH_DENY NO	Permits both read and write access to the file.

NOTE: If a new file is created by this function, the share flag is ignored.

See Also

close, **dup**, **dup2**, **open**

Example

sopen

```
#include <errno.h>
#include <fcntl.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <nwbindrv.h>
#include <nwshare.h>

#define BUFFER_SIZE 512

main()
{
   int handle, rc, i, ch = 0;
   char buffer[BUFFER_SIZE];
   errno = 0;
   printf ("Login %s\r\n\n", LoginToFileServer("supervisor",
      OT_USER,"\x0") ? "Failed" : "Succeeded" );
   handle = sopen ("test.c", O_RDWR, SH_DENYNO, 0 );
   if (handle == -1)
   {
      printf ("\r%s\r\n\n",strerror(errno));
      exit (0);
   }
   printf ("\r\n\n***** lock %s\r\n\n",lock(handle,1,10) ?
      "Failed" : "Succeeded");
   printf ("%s\r\n",strerror(errno));
```

```
    printf ("%s\r\n",strerror(errno));
    printf ("NWerror %d\r\n",NetWareErrno);
    getch ();
    printf ("\r\n\n***** unlock %s\r\n\n",unlock(handle,1,10) ?
       "Failed" : "Succeeded");
    printf ("%s\r\n",strerror(errno));
    printf ("NWerror %d\r\n", NetWareErrno);
    Logout ();
}
```

tell

Determines the current file position
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: Other
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <fcntl.h>

LONG tell (
   int   handle);
```

Parameters

handle

(IN) Specifies a file handle.

Return Values

When an error occurs, a value of -1 is returned. Otherwise, the current file position is returned in a system-dependent manner. A value of 0 indicates the start of the file.

If an error occurs, *errno* is set to:

4	EBADF	Bad file number.

Remarks

The *handle* value is the file handle returned by a successful execution of the **open**, **sopen**, **creat**, or **fileno** function. This function can be used in conjunction with **lseek** to reset the current file position.

See Also

close, **creat**, **eof**, **filelength**, **fileno**, **fstat**, **ftell**, **isatty**, **lseek**, **open**, **read**, **sopen**, **write**

Example

tell

```
#include <fcntl.h>

LONG   position;
int    handle;
position = tell (handle);
```

unlock

Unlocks a region of previously locked data in a file
Local Servers: blocking
Remote Servers: blocking
Classification: Other
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <fcntl.h>

int unlock (
   int    handle,
   LONG   offset,
   LONG   length);
```

Parameters

handle

(IN) Specifies a file handle.

offset

(IN) Specifies the starting byte.

length

(IN) Specifies the amount of data (in bytes).

Return Values

unlock returns a value of 0 if successful and a value of -1 when an error occurs.

If an error occurs,*errno* can be set to:

4	EBADF	Bad file number.
19	EWRNGKND	The region was not locked.

If **unlock** does not complete successfully,*NetWareErrno* is set.

Remarks

The **unlock** function unlocks the region of the file previously locked with

the **lock** function that specified the same offset as the call to **unlock**. If the file does not have a locked region starting at *offset*, the unlock function returns a value of -1 and sets *errno* to EWRNGKND. All locked regions of a file should be unlocked before a file is closed.

See Also

dup, **dup2**, **lock**, **open**, **sopen**

write

Writes data (blocks even if writing to the screen)
Local Servers: blocking
Remote Servers: blocking
Classification: POSIX
SMP Aware: No
Service: Operating System I/O

Syntax

```
#include <unistd.h>

LONG write (
   int    handle,
   char   *buffer,
   LONG   len);
```

Parameters

handle

(IN) Specifies a file handle.

buffer

(IN) Point to the address at which to start transmitting data.

len

(IN) Specifies the number of bytes transmitted.

Return Values

write returns the number of bytes of data transmitted to the file. When there is no error, this is the number given by the*len* parameter. In the case of an error, such as there being no space available to contain the file data, the return value is less than the number of bytes transmitted. A value of -1 is returned in the case of output errors.

If an error occurs,*errno* can be set to:

4	EBADF	Bad file number.

If **write** does not complete successfully,*NetWareErrno* is set.

Remarks

This function also works on the DOS partition.

The *handle* value is returned by **open**, **sopen**, or **creat**. The access mode must have included either O_WRONLY or O_RDWR when **open** or **sopen** was invoked. The data is written to the file at the end when the file was opened with O_APPEND included as part of the access mode; otherwise, it is written at the current file position for the file in question. This file position can be determined with **tell** and can be set with **lseek**.

UNIX Streams

For Stream files, the operation of **write** is determined by the values of the minimum and maximum n-byte range (packet size) accepted by the Stream. These values are contained in the topmost Stream module. Unless the user pushes the topmost module, these values cannot be set or tested from user level.

If n-byte falls within the packet size range, n-byte bytes are written.

If n-byte does not fall within the range and the minimum packet size value is zero, **write** breaks the buffer into maximum packet size segments prior to sending the data downstream (the last segment can contain less than the maximum packet size).

If nbyte does not fall within the range and the minimum value is nonzero, **write** fails with *errno* set to ERANGE. Writing a zero-length buffer (n-byte is 0) sends zero bytes with zero bytes returned.

For Stream files, if O_NDELAY is not set and the Stream cannot accept data (the stream write queue is full due to internal flow control conditions), **write** blocks until data can be accepted. O_NDELAY prevents a process from blocking due to flow control conditions. If O_NDELAY is set and the Stream cannot accept data, **write** fails. If O_NDELAY is set and part of the buffer has been written when a condition in which the Stream cannot accept additional data occurs, **write** terminates and returns the number of bytes written.

See Also

close, **creat**, **dup**, **dup2**, **eof**, **filelength**, **fileno**, **fstat**, **isatty**, **lseek**, **open**, **read**, **sopen**, **tell**

Example

write

```
#include   <stddef.h>
#include   <unistd.h>
#include   <fcntl.h>
#include   <errno.h>
#include   <string.h>
```

```
main()
{
   int fh,size;
   char buffer[] = { "NLM test file" };
   size = strlen(buffer);
   errno = 0;
   if((fh = creat ("test.dt1",0)) != -1)
   {
      printf ("open 1:  %s\r\n",strerror(errno));
   }
   else
   {
      if(write (fh,buffer,size) < size)
      printf ("write 1:  %s\r\n",strerror(errno));
      printf ("file length 1: %d\r\n",filelength(fh));
      if(close (fh) < 0)
      printf ("close 1:  %s\r\n",strerror(errno));
   }
   if((fh = creat ("test.dt2",0)) != -1)
   {
      printf ("open 2:  %s\r\n",strerror(errno));
   }
   else
   {
      if(write (fh,buffer,size) < size)
      printf ("write 2:  %s\r\n",strerror(errno));
      printf ("file length 2: %d\r\n",filelength(fh));
      if(close (fh) < 0)
      printf ("close 2:  %s\r\n",strerror(errno));
   }
   if((fh = creat ("test.dt3",0)) != -1)
   {
      printf ("open 3:  %s\r\n",strerror(errno));
   }
   else
   {
      if(write (fh,buffer,size) < size)
      printf ("write 3:  %s\r\n",strerror(errno));
      printf ("file length 3: %d\r\n",filelength(fh));
      if(close (fh) < 0)
      printf ("close 3:  %s\r\n",strerror(errno));
   }
   printf ("\r\n exit:  %s\r\n",strerror(errno));
   getch ();
}
```

Path and Drive

Path and Drive: Guides

Path and Drive: Task Guide

Path and Drive: Concept Guide

Path and Drive: Tasks

Listing Network Drives

The following steps allow you to determine if the specified drive is a NetWare® drive.

1. **Initialize the client libraries by calling NWCallsInit.**
2. **For each of the drives, 1 through 26, call NWGetDriveStatus and check the *status* parameter to determine if the drive is a NetWare drive.**

Listing Network Drives: Example

Parent Topic:

Path and Drive: Guides

Mapping Network Drives

This task allows you to associate a NetWare® path with a client's drive.

1. **Determine the path to be mapped to and the drive letter that is to be associated with the path.**
2. **Call NWGetDriveStatus to determine if the specified drive is available as a network drive.**
3. **Call NWParsePath to determine if a connection exists to the server specified in the path.**
4. **If a connection does not exist to the specified server, establish a connection.**
5. **Remove the server name from the path by calling NWStripServerOffPath.**
6. **Map the drive by calling NWSetDriveBase.**

Mapping a Network Drive: Example

Parent Topic:

Path and Drive: Guides

Path and Drive: Examples

Listing Network Drives: Example

The following code lists a station's network drives. The program loops through the 26 drives calling **NWGetDriveStatus** to find those mapped to the network. The program doesn't check the additional six drives available under Netx.

Listing Network Drives for a Workstation

```
#include   <stdio.h>
#include   <stdlib.h>
#include   <nwdpath.h>
#include   <nwmisc.h>

void main(void)
{
   NWCCODE   ccode;
   nuint16   status;
   nstr8     rootPath[304];
   nstr8     relPath[304];
   nuint16   drive;

   ccode = NWCallsInit(NULL, NULL);
   if(ccode)
      exit(1);

   for(drive = 1; drive <= 26; drive++)
   {
      ccode = NWGetDriveStatus(drive, NW_FORMAT_SERVER_VOLUME,
               &status, NULL, rootPath,
               relPath, NULL);
      if(ccode != NW_INVALID_DRIVE)
      {
         if(ccode)
         {
            puts("Unable to get drive mappings.\n");
            exit(1);
         }
         if(status & NW_NETWARE_DRIVE)
         {
            printf("Drive %c: = %s \\", drive + '@',
                    rootPath);
            puts(relPath);
         }
```

```
        }
    }
}
```

Parent Topic:

Path and Drive: Guides

Mapping Network Drives: Example

The following code calls **NWSetDriveBase** to map a network drive. Command line parameters correspond to the drive letter and directory path (including a server name). **NWGetDriveStatus** determines whether the drive is eligible. **NWParsePath** determines whether a connection exists to the specified server.

Notice that if there is no connection to the server and Directory Services is available, the code attempts to establish an authenticated connection.

The volume path is obtained by calling **NWStripServerOffPath**, which must be passed separately to **NWSetDriveBase**. Before mapping the drive, the program checks whether the associated connection has been authenticated. Errors are returned if the connection hasn't been authenticated either through Directory Services or bindery login.

Mapping a Network Drive

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <nwnet.h>
#include <nwlocale.h>
#include <nwdpath.h>
#include <nwmisc.h>
#include <nwserver.h>
#include <nwerror.h>
#include <nwdsasa.h>
#include <nwndscon.h>
#include <unicode.h>

void main(int argc, char *argv[])
{
   NWDS_Session_Key_T   sessionKey;
   NWCONN_HANDLE        conn;
   NWCCODE              ccode, DSIsAvailable;
   nuint16              driveNum, status;
   LCONV                lconvInfo;
   char N_FAR           *countryPtr;
   nstr8                serverName[50], driveLetter;
   nstr8                *volumePath;
```

```
if(argc < 2)
{
   printf("Usage:   MAP <driveLetter> <directoryPath>\n");
   printf("Example: MAP K DSSERVER1/SYS:\n");
   exit(1);
}
ccode = NWCallsInit(NULL, NULL);
if(ccode)
   exit(1);

driveLetter = (char)toupper(argv[1][0]);
driveNum = driveLetter - '@';

ccode = NWGetDriveStatus(driveNum, 0, &status, NULL, NULL, NULL, NUL
if(ccode)
{
   if(ccode == NW_INVALID_DRIVE)
      printf("%c is an invalid drive.\n", driveLetter);
   else
      printf("Unable to find drive status.\n");
   exit(1);
}
if(status != NW_UNMAPPED_DRIVE)
{
   printf("Drive %c is in use.\n", driveLetter);
   exit(1);
}

ccode = NWParsePath(strupr(argv[2]), serverName, &conn, NULL, NULL);
if((ccode == NO_CONNECTION_TO_SERVER))
{
   DSIsAvailable = NWIsDSAuthenticated( );
   if (!DSIsAvailable)
   {
      printf("You are not logged into %s\n", serverName);
      exit(1);
   }
   else
   {
      ccode = NWAttachToFileServer(serverName, 0, &conn);
      if(ccode)
      {
         printf("Could not attach to file server %s\n",
                  serverName);
         exit(1);
      }
      /* Set up locale information */
      countryPtr = NWLsetlocale(LC_ALL, "");
      if(countryPtr == NULL)
      {
         printf("NWLsetlocale failed.\n");
         exit(1);
```

```
            exit(1);
         }
         /* Initialize the LCONV structure to the locale information */
         NWLlocaleconv(&lconvInfo);

         ccode = NWInitUnicodeTables(lconvInfo.country_id,
                                     lconvInfo.code_page);
         if (ccode)
            exit(1);

         ccode = NWDSAuthenticate(conn, 0L, &sessionKey);
         if(ccode)
         {
            printf("Error authenticating to server\n");
            NWFreeUnicodeTables( );
            exit(1);
         }
         NWFreeUnicodeTables( );
      }
   }
   volumePath = NWStripServerOffPath((pnstr8)argv[2], NULL);
   ccode = NWSetDriveBase(driveNum, conn, 0, volumePath, 0);
   if(ccode)
   {
      printf("Error. Drive not mapped\n");
      exit(1);
   }
   printf("Drive %c: = %s\n", driveLetter, volumePath);
}
```

Parent Topic:

Path and Drive: Guides

Path and Drive: Concepts

Network Drive Functions

These functions map network drives, return drive information, perform parsing on path strings, and access the Netx search drive vector. It is possible that a specific client supports only a subset of these functions. See Path and Drive: Functions for details on client support.

Function	Comment
NWDeleteDriveBase	Deletes a network drive mapping.
NWGetDriveInformation	Returns information about the specified drive.
NWGetDriveStatus	Returns the status of the specified drive and, optionally, the associated connection and its path in various formats.
NWGetFirstDrive	Returns the first non-local drive.
NWGetPathFromDirectoryEntry	Returns the path name from an entry in a NetWare® server's directory entry table. This function is for NetWare 2.2.
NWGetSearchDriveVector	Returns the shell's search drive vector. If the Requester is running at the workstation, netx.vlm must be loaded for this call to succeed.
NWParseNetWarePath	Parses a path and returns the connection handle, directory handle, and new path to be used by subsequent NetWare requests.
NWParsePath	Parses a path string.
NWSetDriveBase	Maps the target drive to the specified directory path.
NWSetInitDrive	Sets the initial drive on the specified NetWare server. This function is for an OS/2 workstation.
NWSetSearchDriveVector	Sets the shell's search drive vector. If the Requester is running at the workstation, netx.vlm must be loaded for this function to succeed.
NWStripServerOffPath	Parses a server or volume path, copies the server name to the buffer specified by

	server, and returns a pointer to the volume path.

Parent Topic:

Path and Drive: Guides

Network Drive Mappings

An OS/2 or DOS Requester workstation can have up to 26 network drive mappings. The first 26 drives are named alphabetically A through Z. The drives are identified numerically (1 = A, 2 = B, etc.).

Parent Topic:

Path and Drive: Guides

Related Topics:

Mapping Network Drives

Mapping a Network Drive: Example

Path and Drive Introduction

Path and Drive services control the workstation's relationship to the network. Specifically, they configure the workstation environment by managing network drive mappings. However, these services don't formulate requests for NetWare® servers.

Path and Drive services include functions that manage network drive mappings. The following are the most commonly used.

NWGetDriveStatus returns information about a drive mapping.

NWSetDriveBase sets a drive mapping.

NWDeleteDriveBase deletes a drive mapping.

For a complete list of related functions, see Network Drive Functions.

When working with drive mappings, you must express all path parameters in a manner consistent with the local name space. For example, all DOS file path parameters must be in upper case.

Parent Topic:

Path and Drive: Guides

Path and Drive: Functions

ConvertNameToFullPath

Converts a path to an absolute path specification that includes a volume specification

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 2.x, 3.x, 4.x

Platform: NLM

SMP Aware: Yes

Service: Path and Drive

Syntax

```
#include <stdlib.h>
#include <nwdir.h>

int ConvertNameToFullPath (
   char   *partialPath,
   char   *fullPath);
```

Parameters

partialPath

(IN) Specifies a string containing the partial path that is to be converted to a complete path.

fullPath

(OUT) Specifies the buffer where the complete path is to be returned (maximum 255 characters).

Return Values

0 (0x00)	ESUCCESS: Only fails if the *partialPath* parameter is not valid.
22 (0x16)	EBADHNDL

Remarks

ConvertNameToFullPath accepts a filename, or any relative or absolute path, and returns the absolute path (including a volume specification).

Call **ConvertNameToFullPath** when a user is entering a filename (which may or may not be entered as a full path specification) and you want a

full path specification to open the file.

ConvertNameToFullPath uses **ParsePath** to construct the *fullPath* parameter string.

See Also

ConvertNameToVolumePath, **ParsePath**, and File System

ConvertNameToVolumePath

Converts a path to an absolute path specification that does not include the volume specification

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 2.x, 3.x, 4.x

Platform: NLM

SMP Aware: Yes

Service: Path and Drive

Syntax

```
#include <nwdir.h>

int ConvertNameToVolumePath (
   char   *fileName,
   char   *path);
```

Parameters

fileName

(IN) Specifies the name of the file that is to be converted to a complete path from the volume.

path

(OUT) Specifies the buffer where the complete path is to be returned (maximum 255 characters).

Return Values

0	(0x00)	ESUCCESS
22	(0x16)	EBADHNDL

Remarks

ConvertNameToVolumePath accepts a filename, or any relative or absolute path, and returns the absolute path (not including a volume specification). The volume name is not included in the path.

Call **ConvertNameToVolumePath** when a user is entering a filename (which may or may not be entered as a full path specification) and you want a full path specification to open the file.

See Also

ConvertNameToFullPath, and File System

NWDeleteDriveBase

Deletes a network drive mapping
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWDeleteDriveBase (
   nuint16   driveNum,
   nuint16   driveScope);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWDeleteDriveBase
  (driveNum : nuint16;
   driveScope : nuint16
) : NWCCODE;
```

Parameters

driveNum

(IN) Specifies the drive number whose mapping is being deleted (A=1, B=2, . . .).

driveScope

Reserved for Novell® use only; must be 0.

Return Values

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

0x0000	SUCCESSFUL
0x8804	BAD_DRIVE_BASE
0x8836	INVALID_PARAMETER
0x883C	NOT_MY_RESOURCE
0x8875	INVALID_DRIVE_NUM

0x89FF	INVALID_DRIVE_NUMBER

Remarks

If *driveNum* is zero (0), the current drive will be deleted if it belongs to the NetWare® OS.

Most operating systems will determine if the path is valid before **NWDeleteDriveBase** returns.

Under DOS and Windows 3.1, 0x0001 Invalid Function will be returned if the path is not valid and NETX is not loaded. If NETX is loaded, 0x8804 BAD_DRIVE_BASE will be returned.

Under Windows95, 0x0003 Path Not Found will be returned if the path is invalid.

Under Windows NT, INVALID_PARAMETER will be returned if an unmapped drive is being referenced. INVALID_DRIVE_NUM will be returned if an invalid drive number is being used.

Under NLM, INVALID_SHELL_CALL is always returned.

NCP Calls

None

See Also

NWSetDriveBase

NWGetDirBaseFromPath

Gets a volume number, a directory base for the specified name space, and a directory base for the DOS name space entry

Local Servers: blocking

Remote Servers: blocking

Platform: NLM

Classification: 3.12, 4.x

SMP Aware: No

Service: Name Space

Syntax

```
#include <nwfile.h>

N_EXTERN_LIBRARY( NWCCODE )NWGetDirBaseFromPath (
   char   *path,
   BYTE   nameSpace,
   LONG   *volNum,
   LONG   *NSDirBase,
   LONG   *DOSDirBase);
```

Parameters

path

(IN) Specifies the directory path to generate a directory base (number) for.

nameSpace

(IN) Specifies the name space to generate the directory base (number) for.

volNum

(OUT) Receives the volume number that corresponds with *path*.

NSDirBase

(OUT) Receives a directory index for the specified name space.

DOSDirBase

(OUT) Receives a directory index for the DOS name space of the entry.

Return Values

If successful, this function returns zero. Otherwise, it returns a nonzero error code.

Remarks

This function gets a volume number, a directory base for the specified name space, and a directory base for the DOS name space for the entry.

NWGetDriveInformation

Returns information about the specified drive
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetDriveInformation (
   nuint16               driveNum,
   nuint16               mode,
   NWCONN_HANDLE N_FAR  *conn,
   NWDIR_HANDLE N_FAR   *dirHandle,
   pnuint16              driveScope,
   pnstr8                dirPath);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWGetDriveInformation
  (driveNum : nuint16;
   mode : nuint16;
   Var conn : NWCONN_HANDLE;
   Var dirHandle : NWDIR_HANDLE;
   driveScope : pnuint16;
   dirPath : pnstr8
) : NWCCODE;
```

Parameters

driveNum

(IN) Specifies the drive number for which to get the status (A=1, B=2, C=3, . . .); pass 0 for current drive.

mode

Currently unused.

conn

(OUT) Points to the connection ID of the server the drive is currently mapped to.

dirHandle

(OUT) Points to the directory handle associated with the specified drive.

driveScope

(OUT) Points to the drive scope (currently returns GLOBAL).

dirPath

(OUT) Points to the current directory of the specified drive.

Return Values

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

0x0000	SUCCESSFUL
0x000F	DOS_INVALID_DRIVE
0x883C	NOT_MY_RESOURCE
0x89FF	INVALID_DRIVE_NUMBER

Remarks

If *driveNum* is 0, information about the current drive is returned.

Under OS/2, **NWGetDriveInformation** calls the NetWare IFS to retrieve information and then return it. INVALID_DRIVE_NUMBER is returned if *driveNum* is greater than 26. NOT_MY_RESOURCE is returned if the specified drive is not a NetWare drive.

DOS_INVALID_DRIVE is returned if the drive is not defined.

Under DOS, several Shell functions are called to determine the current drive, the connection ID, the directory handle, and the path. INVALID_DRIVE_NUMBER is returned if *driveNum* is greater than 31. NOT_MY_RESOURCE is returned if the specified drive is not a NetWare drive.

If VLMs are running, *dirHandle* returns 0. VLMs do not associate a directory handle with a mapped drive, no directory handle can be returned. For example, if NETX version 3.32 is running, **NWGetDriveInformation** will return a valid *dirHandle* (non-zero) and a valid *dirPath*. If VLM version 1.20 is running, **NWGetDriveInformation** returns a *dirHandle* of zero and a valid *dirPath* (the same *dirPath* returned when NETX was running).

Under Windows NT, a *dirHandle* will not be returned. Under all other platforms, if *dirHandle* does not point to NULL, a*dirHandle* will be returned if NETX support is available. Otherwise, **NWGetDriveInformation** will return NWE_REQUESTER_FAILURE (0x88FF).

Under NLM, INVALID_SHELL_CALL is always returned.

NCP Calls

None

See Also

NWGetFirstDrive

NWGetDriveStatus

Returns the status of the specified drive and, optionally, the associated connection and its path in various formats

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

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetDriveStatus (
   nuint16                driveNum,
   nuint16                pathFormat,
   pnuint16               status,
   NWCONN_HANDLE N_FAR   *conn,
   pnstr8                 rootPath,
   pnstr8                 relPath,
   pnstr8                 fullPath);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWGetDriveStatus
  (driveNum : nuint16;
   pathFormat : nuint16;
   status : pnuint16;
   Var conn : NWCONN_HANDLE;
   rootPath : pnstr8;
   relPath : pnstr8;
   fullPath : pnstr8
) : NWCCODE;
```

Parameters

driveNum

(IN) Specifies the drive number for which to get the status (A=1, B=2, C=3, . . .); pass 0 for current drive.

pathFormat

(IN) Specifies the desired format for the return paths.

status

(OUT) Points to a bit mask indicating if the drive is local and/or

networked.

conn

(OUT) Points to the connection handle of the path *driveNum* is mapped to, if any (optional).

rootPath

(OUT) Points to the base path *driveNum* is mapped to (optional).

relPath

(OUT) Points to the path (relative to the *rootPath* parameter) to which the drive number is mapped (optional).

fullPath

(OUT) Points to the full path of *driveNum*, if it is a network drive (optional).

Return Values

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

0x0000	SUCCESSFUL
0x000F	NW_INVALID_DRIVE
0x8800	Unknown Error Occurred; Unable to Complete Request
0x883C	NOT_MY_RESOURCE

Remarks

Currently, **NWGetDriveStatus** returns the status of local drives, but does not return path strings for these paths to prevent critical errors from occurring on removable drives. (May change with future releases.)

pathFormat expects one of the following four constants:

```
NW_FORMAT_NETWARE          0
NW_FORMAT_SERVER_VOLUME    1
NW_FORMAT_DRIVE            2
NW_FORMAT_UNC                3
```

For the NetWare, Server Volume, and UNC constants, the value of the *fullPath* parameter will equal the value of the *rootPath* parameter, plus a backslash character (`\'), plus the value of the *relPath* parameter. For the Drive constant, the value of the *fullPath* parameter will equal the value of the *rootPath* parameter plus the value of the *relPath* parameter (without adding a backslash character).

The following tables explain what will be returned in each of the path output parameters for each of the *pathFormat* constants.

Assume you are in dir2 and drive letter Q is root mapped to the following:

```
server\volume:dir1
```

	rootPath	*relPath*	*fullPath*
NetWare	volume:dir1	dir2	volume:dir1\dir2
Server Volume	server\volume:dir1	dir2	server\volume:dir1\dir2
Drive	Q:\	dir2	Q:\dir1\dir2
UNC	\\server\volume\dir1	dir2	\\server\volume\dir1\dir2

Assume you are in dir1\dir2 and drive letter Q is root mapped to the following:

```
server\volume:
```

	rootPath	*relPath*	*fullPath*
NetWare	volume:	dir1\dir2	volume:\dir1\dir2
Server Volume	server\volume:	dir1\dir2	server\volume:\dir1\dir2
Drive	Q:\	dir1\dir2	Q:\dir1\dir2
UNC	\\server\volume	dir1\dir2	\\server\volume\dir1\dir2

status returns a bit mask indicating if a drive is a local and/or networked drive:

C Value	**Pascal Value**	**Value Name**
0x0000	$0000	NW_UNMAPPED_DRIVE

0x0000	$0000	NW_FREE_DRIVE
0x0400	$0400	NW_CDROM_DRIVE
0x0800	$0800	NW_LOCAL_FREE_DRIVE
0x1000	$1000	NW_LOCAL_DRIVE
0x2000	$2000	NW_NETWORK_DRIVE
0x4000	$4000	NW_PNW_DRIVE
0x8000	$8000	NW_NETWARE_DRIVE

NW_LOCAL_DRIVE indicates the specified drive letter is lower than the first networked drive which usually defaults to F: and is set in the net.cfg file.

Under NLM, INVALID_SHELL_CALL is always returned.

NCP Calls

None

See Also

NWGetFirstDrive

NWGetFirstDrive

Returns the first non-local drive
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetFirstDrive (
   pnuint16   firstDrive);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWGetFirstDrive
  (firstDrive : pnuint16
) : NWCCODE;
```

Parameters

firstDrive

(OUT) Points to the first non-local drive (A=1, B=2, C=3. . .).

Return Values

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

0x0000	SUCCESSFUL
0x000F	Unknown error occurred

Remarks

If an unknown error occurs while obtaining drive information, **NWGetFirstDrive** returns 0x000F; this is very rare.

Under NLM, INVALID_SHELL_CALL is always returned.

NCP Calls

None

See Also

NWGetDriveStatus

NWGetPathFromDirectoryBase

Returns the path name from an entry in the directory entry table for a NetWare server

NetWare Server: 3.x, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY (NWCCODE) NWGetPathFromDirectoryBase (
   NWCONN_HANDLE   conn,
   nuint8          volNum,
   nuint32         dirBase,
   nuint8          namSpc,
   pnuint8         len,
   pnstr8          pathName);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWGetPathFromDirectoryBase
  (conn : NWCONN_HANDLE;
   volNum : nuint8;
   dirBase : nuint32;
   namSpc : nuint8
   len : pnuint8;
   pathName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number.

dirBase

(IN) Specifies the directory entry number in the name space specified by the *namSpc* parameter.

namSpc

(IN) Specifies the name space used by the directory entry number.

len

(OUT) Points to the path length and specifies how much of the buffer pointed to by the *pathName* parameter was used.

pathName

(OUT) Points to the buffer containing the path name (at least 256 characters).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x899C	INVALID_PATH

Remarks

NWGetPathFromDirectoryBase maps a directory entry number to a path under a specified name space. The path is returned as a group of components. Each directory, subdirectory, or file in the path is considered to be a component. Each component is length preceeded and followed by the next component. The path is NULL-terminated.

The *namSpc* parameter can have the following values:

0 NW_NS_DOS
1 NW_NS_MAC
2 NW_NS_NFS
3 NW_NS_FTAM
4 NW_NS_LONG

You must allocate memory for the buffer pointed to by the *pathName* parameter.

NCP Calls

0x2222 23 243 Map Directory Number to Path

See Also

NWGetPathFromDirectoryEntry

NWGetPathFromDirectoryEntry

Returns the path name from an entry in a NetWare server's directory entry table

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWGetPathFromDirectoryEntry (
   NWCONN_HANDLE   conn,
   nuint8          volNum,
   nuint16         dirEntry,
   pnuint8         len,
   pnstr8          pathName);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWGetPathFromDirectoryEntry
  (conn : NWCONN_HANDLE;
   volNum : nuint8;
   dirEntry : nuint16;
   len : pnuint8;
   pathName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number.

dirEntry

(IN) Specifies the directory entry number.

len

(OUT) Points to the path length and specifies how much of the buffer

pointed to by the *pathName* parameter was used.

pathName

(OUT) Points to the buffer containing the path name (at least 256 characters).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

NCP Calls

0x2222 22 26 Get Path Name Of A Volume---Directory Number Pair

See Also

NWScanOpenFilesByConn2

NWGetSearchDriveVector

Returns the shell's search drive vector
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, Windows 3.1, Windows NT, Windows95
Service: Path and Drive

Syntax

```
#include<nwcaldef.h>
#include<nwdpath.h>
or
#include<nwcalls.h>

N_EXTERN_LIBRARY( NWCCODE )NWGetSearchDriveVector (
   pnstr8 vectorBuffer);
```

Pascal Syntax

```
Function NWGetSearchDriveVector
  (vectorBuffer : pnstr8
) : NWCCODE;
```

Parameters

vectorBuffer

(OUT) Points to a 16-byte buffer receiving the search drive vectors.

Return Values

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

0x0000	SUCCESSFUL
0x8811	INVALID_SHELL_CALL
0x89FF	NWE_REQUESTER_FAILURE

Remarks

NWGetSearchDriveVector will only return a drive vector if either NETX.COM or NETX.EXE is running. Otherwise, **NWGetSearchDriveVector** returns 0xFF in *vectorBuffer*.

Each byte of the vector is a drive handle. The vector list is terminated by a

byte (0xFF).

Under NLM, INVALID_SHELL_CALL is always returned.

NCP Calls

None

NWParseNetWarePath

Parses a path and returns the connection handle, directory handle, and new path to be used by subsequent NetWare requests

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWParseNetWarePath (
   pnstr8                 path,
   NWCONN_HANDLE N_FAR   *conn,
   NWDIR_HANDLE N_FAR    *dirHandle,
   pnstr8                newPath);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWParseNetWarePath
  (path : pnstr8;
   Var conn : NWCONN_HANDLE;
   Var dirHandle : NWDIR_HANDLE;
   newPath : pnstr8
) : NWCCODE;
```

Parameters

path

(IN) Points to the path being parsed in capital letters.

conn

(OUT) Points to the NetWare server connection handle.

dirHandle

(OUT) Points to the directory handle.

newPath

(OUT) Points to the new path, relative to the directory handle; this parameter should be a buffer of at least 256 characters.

Return Values

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

0x0000	SUCCESSFUL
0x880F	NO_CONNECTION_TO_SERVER
0x883C	NOT_MY_RESOURCE

Remarks

NWParseNetWarePath does not check the validity of any volume or directory names in the path string.

path must be in capital letters or calling **NWParseNetWarePath** will fail.

If the path to be parsed is relative to the current directory, **NWParseNetWarePath** assumes the current drive and path so a complete path specification is returned. If the path is on a local drive, **NWParseNetWarePath** returns NOT_MY_RESOURCE. If the path specifies a NetWare server name and there are no connections to that NetWare server, NO_CONNECTION_TO_SERVER is returned.

Under OS/2, **NWParseNetWarePath** returns a directory handle and a path relative to it.

Under DOS, **NWParseNetWarePath** returns zero (0) for the directory handle and a volume:path.

NWParseNetWarePath returns a connection handle under OS/2 and DOS.

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

NWParsePath

NWParsePath

Parses a path string
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWParsePath (
   pnstr8                path,
   pnstr8                serverName,
   NWCONN_HANDLE N_FAR  *conn,
   pnstr8                volName,
   pnstr8               dirPath);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWParsePath
  (path : pnstr8;
   serverName : pnstr8;
   Var conn : NWCONN_HANDLE;
   volName : pnstr8;
   dirPath : pnstr8
) : NWCCODE;
```

Parameters

path

(IN) Points to the path to be parsed.

serverName

(OUT) Points to the server name (48 characters, optional).

conn

(OUT) Points to the connection handle of the server (optional).

volName

(OUT) Points to the volume name (17 characters, optional).

dirPath

dirPath

(OUT) Points to the directory portion of the path; this parameter should be a buffer of at least 256 characters.

Return Values

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

0x0000	SUCCESSFUL
0x880F	NO_CONNECTION_TO_SERVER

Remarks

If the path to be parsed is relative to the current directory, **NWParsePath** assumes the current drive and path so a complete path specification is returned.

```
IF k: is the current drive
AND \dir1 is the current directory on k:
AND dir2 is a directory in dir1
THEN calling NWParsePath with path pointing to "dir2" will cause dirPat
```

If the path to be parsed contains a map rooted drive, *dirPath* will be set to the complete directory path from the volume level.

```
IF k:is map rooted to server1/sys:dir1\
AND dir2 is a directory in dir1
THEN calling NWParsePath with path pointing to "k:dir2" will cause dirP
```

If the path to be parsed is relative to the current directory, the entire directory path will be returned, **without** a preceding `\' character.

```
IF k: is mapped to server1/sys:
AND the current directory path for k: is dir1
AND dir2 is a directory in dir1
THEN calling NWParsePath with path pointing to "k:dir2" will cause dirP
```

If the path to be parsed is on the root directory, *dirPath* will return with a preceding `\' character even if one is not included in the call. This is the only case that will return a preceding `\' character.

```
IF k: is mapped to server1/sys:
AND the current directory path on k: is the root
AND dir1 is a directory on the root
THEN calling NWParsePath with path pointing to "k:dir1" will cause dirP
 local drives and mapped drives.
```

serverName, *conn*, *volName*, and *dirPath* are optional. Substitute NULL if no

returns are desired. However, all parameter positions must be filled.

If the path is on a local drive, return information is placed in the return parameters as follows:

```
serverName    zero-length string
conn          0
volName       drive letter
dirPath       directories from drive letter
```

NWParsePath does not guarantee the path actually exists.

If the path specifies a NetWare server name and there are no connections to that NetWare server, NO_CONNECTION_TO_SERVER is returned. The path specification can be any of the following:

Specification	Function
drive:path	Drive letter is used to determine the network information, if any.
vol:path	Volume and path will be assumed to be relative to the default server.
server vol:path	Information is copied to the associated return buffers and, if requested, the connection handle is obtained using the server name.
path	Current drive is used to determine all the information.

If a map rooted drive is used, *dirPath* will be set to the complete directory path from the volume level.

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

NWParseNetWarePath

NWSetDriveBase

Maps the target drive to the specified directory path
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetDriveBase (
   nuint16          driveNum,
   NWCONN_HANDLE    conn,
   NWDIR_HANDLE     dirHandle,
   pnstr8           dirPath,
   nuint16          driveScope);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWSetDriveBase
  (driveNum : nuint16;
   conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   dirPath : pnstr8;
   driveScope : nuint16
) : NWCCODE;
```

Parameters

driveNum

(IN) Specifies the drive number of the drive being mapped (0=current, 1=A, 2=B,. . .).

conn

(IN) Specifies the NetWare server connection handle to which the drive is mapped.

dirHandle

(IN) Specifies the directory handle associated with *dirPath*.

dirPath

(IN) Points to the directory path the drive will be mapped to.*dirPath* is relative to *dirHandle,* unless *dirHandle* is 0.

driveScope

Reserved for Novell use only; must be 0.

Return Values

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

0x0000	SUCCESSFUL
0x0055	Duplicate Device (DOS, Windows 3.1): The specified drive is already mapped and VLM (rather than NETX) is loaded
0x8801	INVALID_CONNECTION
0x8802	DRIVE_IN_USE (Windows NT): The drive number is already mapped
0x8803	DRIVE_CANNOT_MAP
0x883C	NOT_MY_RESOURCE: Trying to map a local drive
0x8875	INVALID_DRIVE_NUM
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89FF	INVALID_DRIVE_NUMBER (Windows NT): An invalid drive number is being used

Remarks

If the specified drive number is zero, the current drive will be remapped to the specified path. For other drive numbers, if the target drive is already mapped, the mapping must be deleted by calling **NWDeleteDriveBase** before calling **NWSetDriveBase**. Under DOS and Windows 3.1 with NETX loaded and under OS/2, the drive will be mapped correctly even if it is already mapped.

Under DOS, when using the NETX shell, **NWSetDriveBase** can be used to map the temporary drives from 26 to 31. Temporary drives cannot be mapped by calling **NWSetDriveBase** if the VLM redirector is running.

OS/2 does not allow local drives to be mapped.

Under all platforms, CD-ROM drives cannot be mapped.

The file server name should not be specified in the *dirPath* parameter. Specify the file server name in the *conn* parameter. Under NETX.EXE, the server name can be parsed, but VLMs do not parse out the server name.

Under NLM, INVALID_SHELL_CALL is always returned.

Under NLM, INVALID_SHELL_CALL is always returned.

NCP Calls

None

See Also

NWDeleteDriveBase, **NWGetDriveStatus**

NWSetInitDrive

Sets the initial drive on the specified NetWare server
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetInitDrive (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwdpath.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle on which to set the initial drive.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

NWSetInitDrive is used under OS/2 to set the mapping for drive L, the OS/2 drive containing the system login for attaching to a server.

NWSetInitDrive can be called from all platforms; however, it will only

set the correct drive mapping under OS/2. When called from all other platforms, **NWSetInitDrive** returns SUCCESSFUL without setting the correct drive mapping.

Under NLM, INVALID_SHELL_CALL is always returned.

NCP Calls

None

NWSetSearchDriveVector

Sets the Shell's search drive vector
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Path and Drive

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSetSearchDriveVector (
   pnstr8   vectorBuffer);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWSetSearchDriveVector
  (vectorBuffer : pnstr8
) : NWCCODE;
```

Parameters

vectorBuffer

(IN) Points to the vector buffer.

Return Values

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

0x0000	SUCCESSFUL
0x8811	INVALID_SHELL_CALL
0x89FF	NWE_REQUESTER_FAILURE

Remarks

NWSetSearchDriveVector will only return a drive vector if either NETX.COM or NETX.EXE is running. Otherwise, **NWSetSearchDriveVector** returns 0xFF in *vectorBuffer*.

Each element of the vector is a drive handle. The vector list is terminated by a byte (0xFF).

Under NLM, INVALID_SHELL_CALL is always returned.

Under Windows NT, **NWSetSearchDriveVector** immediately returns INVALID_SHELL_CALL.

NCP Calls

None

NWStripServerOffPath

Parses a server or volume path, copies the server name to the buffer specified by server, and returns a pointer to the volume path

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Path and Drive

Syntax

```
#include <nwdpath.h>
or
#include <nwcalls.h>
N_EXTERN_LIBRARY( pnstr8 )NWStripServerOffPath (
   pnstr8   path,
   pnstr8   server);
```

Pascal Syntax

```
#include <nwdpath.inc>

Function NWStripServerOffPath
  (path : pnstr8;
   server : pnstr8
) : pnstr8;
```

Parameters

path

(IN) Points to a string containing a server volume path.

server

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

Return Values

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

0x0000	*path* passed in was NULL
character pointer	pointer to the volume path

NCP Calls

None

See Also

NWParsePath, **NWParseNetWarePath**

ParsePath

Separates a full path into server, volume, and directory specifications
Local Servers: nonblocking
Remote Servers: blocking
NetWare Server: 2.x, 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: Path and Drive

Syntax

```
#include <stdlib.h>
#include <nwdir.h>

int ParsePath (
   char   *path,
   char   *server,
   char   *volume,
   char   *directories);
```

Parameters

path

(IN) Points to the string containing the path to be parsed and can include a server name (255 character maximum).

server

(OUT) Points to the buffer in which to return the server name (48 character maximum).

volume

(OUT) Points to the buffer in which to return the volume name (16 character maximum).

directories

(OUT) Points to the buffer in which to return the directory specification (255 character maximum).

Return Values

0	(0x00)	ESUCCESS: Fails if an invalid path is passed.
22	(0x16)	EBADHNDL

Remarks

ParsePath parses the given path and separates it into server, volume, and directory specifications. Even if the path is not complete (or it is relative to the current working directory), **ParsePath** returns the complete path specification.

Strings for the *server*, *volume*, and *directories* parameters are always converted to uppercase characters.

See Also

StripFileServerFromPath

SetWildcardTranslationMode

Specifies whether wildcard translation is to take place when parsing pathnames and filenames

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 2.x, 3.x, 4.x

Platform: NLM

SMP Aware: No

Service: Path and Drive

Syntax

```
#include <nwdir.h>

BYTE SetWildcardTranslationMode (
   BYTE   newMode);
```

Parameters

newMode

(IN) Specifies the new translation mode (TRUE or FALSE).

Return Values

Returns the old translation mode.

Remarks

SetWildcardTranslationMode enables (TRUE) or disables (FALSE) translation of the following wildcards when parsing path and filenames:

* asterisk

? question mark

. period

When translation is enabled, the high-order bit is changed for all wildcard characters that are parsed in any subsequent file or directory service function. If the high-order bit is 0, it is set to a value of 1. If the high-order bit is 1, it is set to 0.

NetWare uses its own set of rules to interpret wildcards in pathnames. If the high-order bit of a wildcard character is a 1, NetWare interprets that character as a DOS wildcard (this is called an augmented wildcard) and uses DOS rules for interpretation of that wildcard.

StripFileServerFromPath

Removes the name of the server from a full path specification
Local Servers: nonblocking
Remote Servers: N/A
NetWare Server: 2.x, 3.x, 4.x
Platform: NLM
SMP Aware: No
Service: Path and Drive

Syntax

```
#include <stdlib.h>
#include <nwdir.h>

char *StripFileServerFromPath (
   char   *path,
   char   *server);
```

Parameters

path

(IN) Points to the string containing the path from which to remove the server name.

server

(OUT) Points to the buffer in which to place the stripped server name (48 character maximum).

Return Values

Returns a pointer to a path specification stripped of the server name.

Remarks

StripFileServerFromPath removes the name of the server from a path specification. If the *path* parameter does not include a server specification, **StripFileServerFromPath** returns the original path. If the *path* parameter does include a server specification, the returned value begins with the volume specification.

See Also

ParsePath

Stream I/O

Stream I/O: Guides

Stream I/O: Concept Guide

Stream I/O Introduction

Stream I/O Functions

Additional Link

Stream I/O: Functions

Parent Topic:

File Overview

Stream I/O: Concepts

Stream I/O Functions

Function	Purpose
clearerr	Clears end-of-file error indicators for a stream.
fclose	Closes a stream file.
fcloseall	Closes all open stream files except **stdin**, **stdout**, and **stderr**.
fdopen	Associates a stream with a file handle that represents an open file or device.
feof	Tests the end-of-file indicator for a stream.
ferror	Tests the error indicator for a stream.
fflush	Flushes the output buffer of a stream.
fgetc	Returns the next character from the input stream.
fgetchar	Returns the next character from the input stream pointed to by stdin.
fgetpos	Retrieves the current position of a stream.
fgets	Gets a string of characters from a stream and stores them in an array.
fileno	Returns the file handle for a stream.
flushall	Clears all buffers associated with input streams and writes any buffers associated with output streams.
fopen	Opens a file and associates a stream with it.
fprintf	Writes output to a stream under format control.
fputc	Writes a character to the output stream.
fputs	Writes a string to an output stream.
fread	Reads data from a stream.
freopen	Opens a file and associates a previously opened stream with it.
fscanf	Scans input from a stream under format control.
fseek	Changes the read/write position of a stream.
fsetpos	Sets the current position of a stream.
ftell	Returns the current read/write position of a stream.

fwrite	Writes elements to a stream.
getc	Gets the next character from a stream.
getchar	Gets the next character from stdin.
gets	Gets a string from a stream and stores it in an array.
printf	Writes output to the stream designated by *stdout*.
putc	Writes a character to an output stream.
putchar	Writes a character to an output stream.
puts	Writes a specified character string to an output stream and appends a newline character to the output.
rewind	Sets the stream position indicator to the beginning of the file.
scanf	Scans input from a stream.
setbuf	Associates a buffer with a stream after the stream is open and before it has been read or written to.
setvbuf	Associates a buffer with a stream after the stream is open and before it has been read or written to.
tmpfile	Creates a temporary binary file.
ungetc	Pushes a character back onto the specified input stream.
vfprintf	Writes output to a stream under format control.
vfscanf	Scans input from a stream under format control.
vprintf	Writes output to a stream under format control.
vscanf	Scans input from the stream designated by *stdin*.

Parent Topic:

Stream I/O: Guides

Stream I/O Introduction

NOTE: The streams discussed here are standard files, not to be confused with UNIX STREAMS or STREAMS.

Developed by USL®, the UNIX based STREAMS facility, or mechanism, is a collection of system calls, kernel resources, and kernel utility routines. The STREAMS mechanism creates, uses, and dismantles a Stream, which is a full-duplex processing and data transfer path between a driver in kernel space and a process in user space; a STREAM consists of three basic components: a stream head, stream modules (protocol stacks), and a stream driver. For more information about STREAMS functions, see NetWare STREAMS: Guides.

Stream I/O functions can be used for "standard" read and write file operations. Data can be transmitted as characters, strings, or blocks of memory.

A **stream** is the name given to a second-level file that has been opened for data transmission. When a stream is opened, a pointer to a FILE structure is returned. This pointer is used to reference the stream when other functions are subsequently invoked.

Parent Topic:

Stream I/O: Guides

Stream I/O: Functions

clearerr

Clears the end-of-file and error indicators for a stream (function or macro)
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: Yes
Service: Stream I/O

Syntax

```
#include <stdio.h>

void clearerr (
   FILE   *fp);
```

Parameters

fp
(IN) Points to the file to be cleared.

Return Values

None

Remarks

The **clearerr** function or macro clears the end-of-file and error indicators for the file pointed to by *fp*. These indicators are cleared only when the file is opened or by an explicit call to the **clearerr** or **rewind** functions.

See Also

feof, **ferror**, **perror**

Example

clearerr

```
#include <stdio.h>

main ()
{
   FILE   *fp;
   int     c;
```

```
    fp=fopen("testfile", "wt");
    if (ferror (fp) )
    {                                   /* If error,*/
       clearerr (fp);                   /* clear the error */
       fputc (c, fp);                   /* and retry it */
    }
}
```

fclose

Closes a stream file
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fclose (
   FILE   *fp);
```

Parameters

fp
(IN) Points to the file to close.

Return Values

The **fclose** function returns a value of 0 if the file was successfully closed or nonzero if any errors were detected. When an error has occurred,*errno* is set.

Remarks

This function also works on the DOS partition.

The **fclose** function closes the file pointed to by *fp*. If there is unwritten buffered data for the file, it is written before the file is closed. Unread buffered data is discarded. If the associated buffer was automatically allocated, it is deallocated.

See Also

fcloseall, **fdopen**, **fopen**, **freopen**

fcloseall

Closes all open stream files except stdin, stdout and stderr
Local Servers: blocking
Remote Servers: blocking
Classification: Other
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fcloseall (void);
```

Return Values

The **fcloseall** function returns the number of files that were closed, if no errors were encountered. When an error occurs, EOF is returned.

Remarks

The **fcloseall** function closes all open stream files except stdin, stdout, and stderr. Files closed includes files created (and not yet closed) by **fdopen**, **fopen**, and **freopen**.

See Also

fclose, **fdopen**, **fopen**, **freopen**

Example

fcloseall

```
#include <stdio.h>

main ()
{
   printf ("The number of files closed is %d\n", fcloseall () );
}
```

fdopen

Associates a stream with a file handle that represents an open file or device
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: Other
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

FILE *fdopen (
   int          handle,
   const char   *mode);
```

Parameters

handle

(IN) Specifies a file handle.

mode

(IN) Specifies a file mode.

Return Values

The **fdopen** function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, **fdopen** returns a NULL pointer. When an error has occurred, *errno* is set.

Remarks

The **fdopen** function associates a stream with the file handle, *handle*, which represents an opened file or device. The handle was returned by a **creat** or **open** function. The open mode, *mode*, must match the mode with which the file or device was originally opened.

If *mode* does not match the access flags used in opening the file originally, errno will be set to EINVAL and **fdopen** will fail. This includes the mode accesses read, write, append, and binary. See argument *oflag* for function **open**.

The **fdopen** function opens the file whose name is the string pointed to by *filename*, and associates a stream with it. The argument *mode* points to a string beginning with one of the following sequences:

r	Opens file for reading.
w	Creates file for writing, or truncates to zero length; uses default file translation.
a	Appends; opens or creates text file for writing at end-of-file; uses default file translation.
rb	Opens binary file for reading.
rt	Opens text file for reading.
wb	Creates binary file for writing, or truncates to zero length.
wt	Creates text file for writing, or truncates to zero length.
ab	Appends; opens or creates binary file for writing at end-of-file.
at	Appends; opens or creates text file for writing at end-of-file.
r+	Opens file for update (reading and/or writing); uses default file translation.
w+	Creates file for update, or truncates to zero length; uses default file translation.
a+	Appends; opens or creates file for update, writing at end-of-file; uses default file translation.
r+b	Opens binary file for update (reading and/or writing).
r+t	Opens text file for update (reading and/or writing).
w+b	Creates binary file for update, or truncates to zero length.
w+t	Creates text file for update, or truncates to zero length.
a+b	Appends; opens or creates binary file for update, writing at end-of-file.
a+t	Appends; opens or creates text file for update, writing at end-of-file.
rb+	Opens binary file for update (reading and/or writing).
rt+	Opens text file for update (reading and/or writing).
wb+	Creates binary file for update, or truncates to zero length.
wt+	Creates text file for update, or truncates to zero length.
ab+	Appends; opens or creates binary file for update, writing at end-of-file.
at+	Appends; opens or creates text file for update, writing at end-of-file.

See Also

fopen, freopen, open, sopen

Example

fdopen

This example shows how to reverse the effects of redirecting *stdin*.

```
#include <stdio.h>
int func (char *filepath )
{
   int    fd, stdin_fd;
   char   line[512];
   FILE   *fp;

   stdin_fd = fileno(stdin);   /*save descriptor for 'stdin' */
   fd       = dup(stdin_fd);

   if (fd == -1)
      return -1;               /* failed to duplicate input descriptor

   /* use the duplicated descriptor to redirect input... */
   fp = fdopen (fd, "r");

   if (!fp)
      return -2;               /* failed to open duplicated descriptor

   stdin = freopen (filepath, "r", fp);

   if (!stdin)
      return -3;               /* failed to redirect stream input */

   /* use redirected stream (example)... */
   while (gets(line))
      printf("%s\n", line);

   /* UNDO: now undo the dffects of redirecting input... */
   fclose(stdin);

   stdin = fdopen(stdin_fd, "r");

   if (!stdin)
      return -4;               /* failed to reestablish 'stdin' */

   return 0;
}
```

If *stdin* is redirected by a console command such as

```
LOAD NLM-NAME /(CLIB_OPT)/<filename>
```

you can likewise return the standard input to the keyboard by using the statements following the "UNDO" comment in the example.

feof

Tests the end-of-file indicator for a stream (function or macro)
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: ANSI
SMP Aware: Yes
Service: Stream I/O

Syntax

```
#include <stdio.h>

int feof (
   FILE   *fp);
```

Parameters

fp
(IN) Points to the file to be tested.

Return Values

feof returns nonzero if the EOF indicator is set for*fp*.

Remarks

The **feof** function or macro tests the end-of-file indicator for the file pointed to by *fp*. Because this indicator is set when an input operation attempts to read past the end of the file, **feof** detects the end of the file only after an attempt is made to read beyond the end of the file. Thus, if a file contains 10 lines, **feof** does not detect the end of the file after the tenth line is read; it detects the end of the file once the program attempts to read more data.

See Also

clearerr, **ferror**, **fopen**, **freopen**, **read**

Example

feof

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

```
main ()
{
   FILE *fp;
   char buffer[100];
   fgets (buffer, sizeof (buffer), fp);
   while (! feof (fp) )
   {
      process_record (buffer);
      fgets (buffer, sizeof (buffer), fp);
   }
}
```

ferror

Tests the error indicator for a stream (function or macro)
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: ANSI
SMP Aware: Yes
Service: Stream I/O

Syntax

```
#include <stdio.h>

int ferror (
   FILE   *fp);
```

Parameters

fp
(IN) Points to to the file to be tested.

Return Values

ferror returns nonzero if the error indicator is set for *fp*.

Remarks

The **ferror** function or macro tests the error indicator for the file pointed to by *fp*.

See Also

clearerr, **feof**, **strerror**

Example

ferror

```
#include <stdio.h>

main ()
{
   FILE    *fp;
   int     c;
   c = fgetc (fp);
```

```
   if (ferror (fp) )
   {                          /* if end-of-file */
      fclose  (fp);           /* close the file */
      c = EOF;
   }
}
```

fflush

Flushes the output buffer of a stream
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fflush (
   FILE   *fp);
```

Parameters

fp
(IN) Points to the file to be flushed.

Return Values

The **fflush** function returns nonzero if a write error occurs, and returns zero otherwise. If an error occurs, *errno* is set.

Remarks

If the file pointed to by *fp* is open for output or update, the **fflush** function causes any unwritten data to be written to the file. If the file pointed to by *fp* is open for input or update, the **fflush** function undoes the effect of any preceding **ungetc** operation on the stream. If the value of *fp* is NULL, all open files are flushed.

See Also

fgetc, **fgets**, **flushall**, **fopen**, **getc**, **gets**, **setbuf**, **setvbuf**, **ungetc**

fgetc

Returns the next character from the input stream
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fgetc (
   FILE   *fp);
```

Parameters

fp
(IN) Points to the file.

Return Values

The **fgetc** function returns the next character from the input stream pointed to by *fp*. If the stream is at end-of-file, the EOF indicator is set and **fgetc** returns EOF. If a read error occurs, the error indicator is set and **fgetc** returns EOF. If an error occurs, *errno* is set.

Remarks

The **fgetc** function gets the next character from the file designated by *fp*. The character is signed.

See Also

fgets, **fopen**, **getc**, **gets**, **ungetc**

Example

fgetc

```
#include <stdio.h>

main ()
{
```

```
   FILE   *fp;
   int    c;
   fp = fopen ("data.fil", "r");
   while ( (c = fgetc (fp) ) != EOF)
      putchar (c);
   fclose (fp);
}
```

fgetchar

Equivalent to **fgetc** with the argument *stdin* (implemented for NetWare® 3.11 and above)

Local Servers: blocking

Remote Servers: blocking

Classification: Other

SMP Aware: No

Service: Stream I/O

Syntax

```
#include <stdio.h>

int fgetchar (void);
```

Return Values

The **fgetchar** function returns the next character from the input stream pointed to by *stdin*. If the stream is at end-of-file, the EOF indicator is set and **fgetchar** returns EOF. If a read error occurs, the error indicator is set and **fgetchar** returns EOF. When an error has occurred, the global variable *errno* contains a value indicating the type of error detected.

Remarks

The **fgetchar** function is equivalent to **fgetc** with the argument *stdin*.

See Also

fgetc, **getc**, **getchar**

Example

fgetchar

```
#include <stdio.h>

main()
{
   int c;
   while( (c = fgetchar()) != EOF )
      putchar(c);
}
```

fgetpos

Stores the current position of a stream in an object
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fgetpos (
   FILE     *fp,
   fpos_t   *pos);
```

Parameters

fp

(IN) Points to the file.

pos

(OUT) Points to the object into which the position of the file is stored.

Return Values

The **fgetpos** function returns a value of 0 if successful. Otherwise, the **fgetpos** function returns a nonzero value. If an error occurs,*errno* is set.

Remarks

The **fgetpos** function stores the current position of the file pointed to by*fp* in the object pointed to by *pos*. The value stored is usable by the **fsetpos** function for repositioning the file to the position it had at the time of the call to **fgetpos**.

See Also

fopen, **fseek**, **fsetpos**, **ftell**

Example

fgetpos

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

```
#include <stdio.h>

main ()
{
   int    completionCode;
   FILE   *fp;
   fpos_t position;
   completionCode = fgetpos (fp, &position);
   completionCode = fsetpos (fp, &position);
}
```

fgets

Gets a string of characters from a stream and stores them in an array
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

char *fgets (
   char     *buf,
   size_t   n,
   FILE     *fp);
```

Parameters

buf

(OUT) Points to the array into which the characters are to be stored.

n

(IN) Specifies the number of characters to read.

fp

(IN) Points to the file to be read.

Return Values

The **fgets** function returns *buf* if successful. NULL is returned if end-of-file is encountered or if a read error occurs. If an error occurs, *errno* is set.

Remarks

The **fgets** function gets a string of characters from the file designated by *fp* and stores them in the array pointed to by *buf*. The **fgets** function stops reading characters when end-of-file is reached, or when a newline character is read, or when *n*-1 characters have been read, whichever comes first. The newline character is not discarded. A NULL character is placed immediately after the last character read into the array.

The **gets** function is similar to **fgets** except that it operates with *stdin*; it has no size argument, and it replaces a newline character with the NULL character.

See Also

fgetc, **fopen**, **getc**, **gets**

Example

fgets

```
#include <stdio.h>

main ()
{
   FILE   *fp;
   char   buffer[80];
   fp = fopen ("data.fil", "r");
   while (fgets (buffer, 80, fp) != NULL)
      fputs (buffer, stdout);
   fclose (fp);
}
```

fileno

Returns the file handle for a stream (function or macro)
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: Other
SMP Aware: Yes
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fileno (
   FILE   *fp);
```

Parameters

fp
(IN) Points to a stream opened with a previous call to **fopen** or **fdopen**.

Return Values

fileno returns the file handle designated by the *fp* parameter.

If an error occurs, *errno* is set to:

4	EBADF	Bad file number.

If **fileno** does not complete successfully, *NetWareErrno* is set.

Remarks

The returned file handle can be used to access the stream with any of the functions that take a handle.

There are two versions of **fileno** in the NetWare API:

The **fileno** macro in STDIO.H does not do any error checking against a bad file pointer.

The **fileno** function checks the handle passed in. If you want to use this function and are including STDIO.H, place a sequence such as the following in your code:

```
#ifdef fileno
#undef fileno
#endif
```

See Also

close, creat, eof, filelength, fdopen, fopen, fstat, isatty, lseek, open, read, sopen, tell, write

flushall

Clears all buffers associated with input streams and writes any buffers associated with output streams

Local Servers: blocking

Remote Servers: blocking

Classification: Other

SMP Aware: No

Service: Stream I/O

Syntax

```
#include <stdio.h>

int flushall (void);
```

Return Values

The **flushall** function returns the number of open streams. If an output error occurs while writing to a file, *errno* is set.

Remarks

The **flushall** function clears all buffers associated with input streams and writes any buffers associated with output streams. A subsequent read operation on an input file causes new data to be read from the associated file or device.

flushall is equivalent to calling the **fflush** function for all open stream files.

See Also

fflush, **fopen**

Example

flushall

```
#include <stdio.h>

main ()
{
   printf ("The number of open files is %d\n", flushall () );
}
```

fopen

Opens a file and associates a stream with it
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

FILE *fopen (
   const char   *filename,
   const char   *mode);
```

Parameters

filename

(IN) Points to the name of the file to be opened.

mode

(IN) Points to the file mode.

Return Values

The **fopen** function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, **fopen** returns NULL. If an error occurs, *errno* is set.

Remarks

This function also works on the DOS partition.

The **fopen** function opens the file whose name is the string pointed to by *filename*, and associates a stream with it. The argument *mode* points to a string beginning with one of the following sequences:

r	Opens file for reading; uses default file translation
w	Creates file for writing, or truncates to zero length; uses default file translation.
a	Appends; opens or creates text file for writing at end-of-file; uses default file translation.

rb	Opens binary file for reading.
rt	Opens text file for reading.
wb	Creates binary file for writing, or truncates to zero length.
wt	Creates text file for writing, or truncates to zero length.
ab	Appends; opens or creates binary file for writing at end-of-file.
at	Appends; opens or creates text file for writing at end-of-file.
r+	Opens file for update (reading and/or writing); uses default file translation.
w+	Creates file for update, or truncates to zero length; uses default file translation.
a+	Appends; opens or creates file for update, writing at end-of-file; uses default file translation.
r+b	Opens binary file for update (reading and/or writing).
r+t	Opens text file for update (reading and/or writing).
w+b	Creates binary file for update, or truncates to zero length.
w+t	Creates text file for update, or truncates to zero length.
a+b	Appends; opens or creates binary file for update, writing at end-of-file.
a+t	Appends; opens or creates text file for update, writing at end-of-file.
rb+	Opens binary file for update (reading and/or writing).
rt+	Opens text file for update (reading and/or writing).
wb+	Creates binary file for update, or truncates to zero length.
wt+	Creates text file for update, or truncates to zero length.
ab+	Appends; opens or creates binary file for update, writing at end-of-file.
at+	Appends; opens or creates text file for update, writing at end-of-file.

Opening a file with read mode (r as the first character in the *mode* argument) fails if the file does not exist or if it cannot be read. Opening a file with append mode (a as the first character in the *mode* argument) causes all subsequent writes to the file to be forced to the current end-of-file, regardless of previous calls to the **fseek** function. When a file is opened with update mode (+ as the second or third character of the *mode* argument), both input and output can be performed on the

associated stream.

NOTE: For an example of how to reverse the effect of redirecting *stdin*, see the example for **fdopen**.

See Also

fclose, **fcloseall**, **fdopen**, **freopen**

Example

fopen

```
#include <stdio.h>

main ()
{
   char   filename[13];
   FILE   *fp;
   strcpy (filename, "REPORTAA.DAT");
   fp = fopen (filename, "r");
}
```

fprintf

Writes output to a stream under format control
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fprintf (
   FILE         *fp,
   const char   *format,
   ... );
```

Parameters

fp

(IN) Points to the file to be written to.

format

(IN) Points to the format control string.

Return Values

The **fprintf** function returns the number of characters written or a negative value if an output error occurred. If an error occurs,*errno* is set.

Remarks

The **fprintf** function writes output to the file pointed to by*fp* under control of the argument *format*. The format string is described under the description of the **printf** function.

See Also

printf, **sprintf**, **vfprintf**

Example

fprintf

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

```
#include <stdio.h>

main ()
{
   char   *weekday = {"Saturday"};
   char   *month = {"April"};
   fprintf (stdout, "%s, %s %d, %d\n", weekday, month, 18, 1991);
}
```

produces the following:

```
Saturday, April 18, 1991
```

fputc

Writes a character to the output stream
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fputc (
   int    c,
   FILE   *fp);
```

Parameters

c

(IN) Specifies the character to be written.

fp

(IN) Points to the output stream.

Return Values

The **fputc** function returns the character written. If a write error occurs, the error indicator is set and **fputc** returns EOF. If an error occurs, *errno* is set.

Remarks

The **fputc** function writes the character specified by the argument *c* to the output stream designated by *fp*.

See Also

fclose, **fgetc**, **fopen**, **fputs**, **putc**, **puts**

Example

fputc

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

```
main ()
{
   FILE    *fp;
   int     c;
   fp = fopen ("data.fil", "r");
   while ( (c = fgetc (fp) ) != EOF)
      fputc (c, stdout);
   fclose (fp);
}
```

fputs

Writes a character string to the output stream
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fputs (
   const char   *buf,
   FILE         *fp);
```

Parameters

buf

(IN) Points to the character string to be written.

fp

(IN) Points to the file

Return Values

The **fputs** function returns nonzero if an error occurs; otherwise, it returns a value of 0. If an error occurs, *errno* is set.

Remarks

The **fputs** function writes the character string pointed to by *buf* to the file designated by *fp*. The terminating NULL character is not written.

See Also

fopen, **fputc**, **putc**, **puts**

Example

fputs

```
#include <stdio.h>

main ()
```

```
main ()
{
   FILE   *fp;
   char   buffer [80];
   fp = fopen ("data.fil", "r");
   while (fgets (buffer, 80, fp) ) != NULL)
      fputs (buffer, stdout);
   fclose (fp);
}
```

fread

Reads data from a stream
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

size_t fread (
   void     *buf,
   size_t   elsize,
   size_t   nelem,
   FILE     *fp);
```

Parameters

buf

(OUT) Points to the location to receive data.

elsize

(IN) Specifies the size (in bytes) of each element.

nelem

(IN) Specifies the number of elements.

fp

(IN) Points to the file to be read.

Return Values

The **fread** function returns the number of complete elements successfully read. This value can be less than the requested number of elements.

Call **feof** and **ferror** to determine whether the end of the file was encountered, or if an input/output error has occurred. If an error occurs, *errno* is set.

Remarks

This function also works on the DOS partition.

The **fread** function reads *nelem* elements of *elsize* bytes each from the file specified by *fp*.

See Also

feof, **ferror**, **fopen**, **read**

Example

fread

The following example reads a simple student record containing binary data. The student record is described by the struct student_data declaration.

```
#include <stdio.h>

struct student_data
{
   int student_id;
   unsigned char marks [10];
};

int   read_data (FILE *fp, struct student_data *p)
{
   return (fread (p, sizeof (*p), 1, fp) );
}
```

freopen

Opens a file and associates a stream with it
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

FILE *freopen (
   const char   *filename,
   const char   *mode,
   FILE         *fp);
```

Parameters

filename

(IN) Points to the name of the file to be opened.

mode

(IN) Points to the file mode.

fp

(IN) Points to the file structure.

Return Values

The **freopen** function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, **freopen** returns NULL. If an error occurs,*errno* is set.

Remarks

The stream located by the *fp* pointer is closed. The **freopen** function opens the file whose name is the string pointed to by *filename*, and associates a stream with it. The stream information is placed in the structure located by the *fp* pointer.

The argument *mode* is described in the description of the **fopen** function.

NOTE: For an example of how to reverse the effect of redirecting *stdin*, see the example for **fdopen**.

See Also

fdopen, fopen

Example

freopen

```
#include <stdio.h>
main ()

{
   FILE    *fp;
   fp = freopen ("report.dat", "r", stdin);
}
```

fscanf

Scans input from a stream under format control
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fscanf (
   FILE          *fp,
   const char    *format,
   ... );
```

Parameters

fp

(IN) Points to the file.

format

(IN) Points to the format control string.

Return Values

The **fscanf** function returns EOF when the scanning is terminated by reaching the end of the input stream. Otherwise, it returns the number of input arguments for which values were successfully scanned and stored. If a file input error occurs, *errno* is set.

Remarks

The **fscanf** function scans input from the file designated by *fp* under control of the argument *format*. Following the format string is a list of addresses to receive values. The format string is described under the description of the **scanf** function.

See Also

scanf, **sscanf**

Example

fscanf

To scan a date in the form "Saturday April 18 1991":

```
#include <stdio.h>

main ()
{
   int    day, year;
   char   weekday[10], month[12];
   FILE   *in_data;
   in_data = fopen ("mydates.dat", "r");
   fscanf (in data, "%s %s %d %d", weekday, month, &day, &year);
}
```

fseek

Changes the read/write position of a stream
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fseek (
   FILE       *fp,
   long int   offset,
   int        where);
```

Parameters

fp

(IN) Points to the file.

offset

(IN) Specifies the file position to seek.

where

(IN) Specifies the relative file position.

Return Values

The **fseek** function returns a value of 0 if successful, nonzero otherwise. If an error occurs, *errno* is set.

Remarks

The **fseek** function changes the read/write position of the file specified by *fp*. This position defines the character to be read or written on the next I/O operation on the file. The argument *fp* is a file pointer returned by **fopen** or **freopen**. The argument *offset* is the position to seek, relative to one of three positions specified by the argument *where*. Allowable values for the *where* parameter are:

SEEK_SET	Relative to beginning of file; the offset must be positive.
SEEK_CU	Relative to the current position in the file.

R	
SEEK_END	Relative to the end of the file.

The **fseek** function clears the end-of-file indicator and undoes any effects of the **ungetc** function on the same file.

Call **ftell** to obtain the current position in the file before changing it. Restore the position by using the value returned by **ftell** in a subsequent call to **fseek** with the *where* parameter set to SEEK_SET.

See Also

fgetpos, **fopen**, **fsetpos**, **ftell**

Example

fseek

You can determine the size of a file by means of the following example, which saves and restores the current position of the file.

```
#include <stdio.h>

long int   filesize (FILE *fp)
{
   long int   save_pos, size_of_file;
   save_pos = ftell (fp);
   fseek (fp, 0L, SEEK_END);
   size_of_file = ftell (fp);
   fseek (fp, save_pos, SEEK_SET);
   return (size_of_file);
}
```

fsetpos

Positions a stream according to the value of an object
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int fsetpos (
   FILE           *fp,
   const fpos_t   *pos);
```

Parameters

fp

(IN) Points to the file.

pos

(IN) Points to the object that specifies the new file position.

Return Values

The **fsetpos** function returns a value of 0 if successful; otherwise, the **fsetpos** function returns a nonzero value. If an error occurs,*errno* is set.

Remarks

The **fsetpos** function positions the file pointed to by*fp* according to the value of the object pointed to by *pos*, which shall be a value returned by an earlier call to the **fgetpos** function on the same file.

See Also

fgetpos, **fopen**, **fseek**, **ftell**

Example

fsetpos

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

```
main ()
{
   FILE     *fp;
   fpos_t   position;
   fgetpos (fp, &position);
   fsetpos (fp, &position);
}
```

ftell

Returns the current read/write position of a stream
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

long int ftell (
   FILE   *fp);
```

Parameters

fp
(IN) Points to the file.

Return Values

The **ftell** function returns the current read/write position of the file specified by *fp*. When an error is detected, a value of -1 is returned. If an error occurs, *errno* is set.

Remarks

The **ftell** function returns the current read/write position of the file specified by *fp*. This position defines the character to be read or written by the next I/O operation on the file. You can use the value returned by **ftell** in a subsequent call to **fseek** in order to set the file to the same position.

See Also

fgetpos, **fopen**, **fseek**, **fsetpos**

Example

ftell

You can determine the size of a file by using the following example, which saves and restores the current position of the file.

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

```
#include <stdio.h>

long int   filesize (FILE *fp)

{
   long int    save_pos, size_of_file;
   save_pos = ftell (fp);
   fseek ( fp, 0L, SEEK_END);
   size_of_file = ftell (fp);
   fseek (fp, save_pos, SEEK_SET);
   return (size_of_file);
}
```

fwrite

Writes elements to a stream
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

size_t fwrite (
   const void   *buf,
   size_t       elsize,
   size_t       nelem,
   FILE         *fp);
```

Parameters

buf

(IN) Points to the buffer containing the data to write.

elsize

(IN) Specifies the size (in bytes) of each element.

nelem

(IN) Specifies the number of elements.

fp

(IN) Points to the file.

Return Values

This function also works on the DOS partition.

The **fwrite** function returns the number of complete elements that are successfully written. This value is less than the requested number of elements only if a write error occurs.

Remarks

The **fwrite** function writes *nelem* elements of *elsize* bytes each to the file specified by *fp*.

See Also

ferror, fopen

Example

fwrite

The following example writes a simple student record containing binary data.

```
#include <stdio.h>

struct student_data
{
   int              student_id;
   unsigned char    marks[10];
};

int write_data (FILE *fp, struct student_data *p)
{
   return (fwrite ( p, sizeof (*p), 1, fp) );
}
```

getc

Gets the next character from a stream (function or macro)
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int getc (
   FILE   *fp);
```

Parameters

fp
(IN) Points to the file.

Return Values

The **getc** function or macro returns the next character from the input stream pointed to by *fp*. If the stream is at end-of-file, the EOF indicator is set and **getc** returns EOF. If a read error occurs, the error indicator is set and **getc** returns EOF. If an error occurs, *errno* is set.

Remarks

The **getc** function or macro gets the next character from the file designated by *fp*. The character is returned as an int value.

The **getc** function is equivalent to **fgetc**, except that it can be implemented as a macro.

See Also

fgetc, **fgets**, **fopen**, **gets**, **ungetc**

Example

getc

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

```
main ()
{
   FILE    *fp;
   int     c;
   fp = fopen ("data.fil", "r");
   while ( (c = getc (fp) ) != EOF)
      putchar (c);
   fclose (fp);
}
```

getchar

Equivalent to **getc** with the argument *stdin* (function or macro)
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int getchar (void);
```

Return Values

The **getchar** function or macro returns the next character from the input stream pointed to by *stdin*. If the stream is at end-of-file, the EOF indicator is set and **getchar** returns EOF. If a read error occurs, the error indicator is set and **getchar** returns EOF. If an error occurs, *errno* is set.

Remarks

The **getchar** function or macro is equivalent to **getc** with the argument *stdin*.

See Also

getc

Example

getchar

```
#include <stdio.h>

main ()
{
   FILE    *fp;
   int      c;
   fp = freopen ("data.fil", "r", stdin);
   while ( (c = getchar () ) != EOF)
      putchar (c);
   fclose (fp);
}
```

gets

Gets a string of characters from *stdin* and stores them in an array
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

char *gets (
   char   *buf);
```

Parameters

buf
(OUT) Points to the array into which the characters are to be stored.

Return Values

The **gets** function returns *buf* if successful. It returns NULL if a read error occurs.

Remarks

The **gets** function gets a string of characters from the file designated by *stdin* and stores them in the array pointed to by *buf* until a newline character is read. Any newline character is discarded, and a NULL character is placed immediately after the last character read into the array.

It is recommended that **fgets** be used instead of **gets** because data beyond the array *buf* is destroyed if a newline character is not read from the input stream *stdin* before the end of the array *buf* is reached.

See Also

fgetc, **fgets**, **fopen**, **getc**, **ungetc**

Example

gets

```
#include <stdio.h>

main ()
{
   char    buffer[80];
   while (gets (buffer) ) != NULL)
      puts (buffer);
}
```

printf

Writes formatted output to a specified file designated by *stdout*
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: Yes
Service: Stream I/O

Syntax

```
#include <stdio.h>

int printf (
   const char   *format,
   ... );
```

Parameters

format

(IN) Points to the format control string.

Return Values

The **printf** function returns the number of characters written, or it returns a negative value if an output error occurred. If an error occurs, *errno* is set.

Remarks

The **printf** function writes output to the file designated by *stdout* under control of the argument *format*.

Format Control String

The format control string consists of ordinary characters, which are written exactly as they occur in the format string, and of conversion specifiers, which cause argument values to be written as they are encountered during the processing of the format string. An ordinary character in the format string is any character, other than a percent (%) character, that is not part of a conversion specifier. A conversion specifier is a sequence of characters in the format string. The conversion specifier begins with a % and is followed, in sequence, by the following:

Zero or more format control flags, which can modify the final effect of the format directive.

An optional decimal integer, or an asterisk character (*), which specifies a minimum field width to be reserved for the formatted item.

An optional precision specification in the form of a period character (.) followed by an optional decimal integer or an asterisk character (*).

An optional type-length specification. It can be any one of the following characters:

h l L N F

A character that specifies the type of conversion to be performed. It can be any one of the following characters:

c d e E f F g G i n o p s u x X

The valid format control flags are:

-	The formatted item is left-justified within the field; normally, items are right-justified.
	A signed, positive object always starts with a plus (+) character; normally, only negative items begin with a sign.
" "	A signed, positive object always starts with a space character; if both + and - are specified, + overrides -.
#	An alternate conversion form is used:
	For o (unsigned octal) conversions, the precision is incremented so that the first digit is 0.
	For x or X (unsigned hexadecimal) conversions, a nonzero value is prepended with 0x or 0X, respectively.
	For e, E, f, g, or G (any floating-point) conversions, the result always contains a decimal-point character, even if no digits follow it; normally, a decimal- point character appears in the result only if there is a digit to follow it.
	In addition to the preceding, for g or G conversions, trailing zeros are not removed from the result.

If no field width is specified, or if the value that is given is less than the number of characters in the converted value (subject to any precision value), a field of sufficient width to contain the converted value is used. If the converted value has fewer characters than are specified by the field width, the value is padded on the left (or right, subject to the left-justification flag) with spaces or zero characters (0). If the field width begins with a zero, the value is padded with zeros; otherwise, the value is padded with spaces. If the field width is *, a value of type int from the argument list is used (before a precision argument or a conversion argument) as the minimum field width. A negative field width value is interpreted as a left- justification flag, followed by a positive field width.

As with the field width specifier, a precision specifier of * causes a value of type int from the argument list to be used as the precision specifier. If no precision value is given, a precision of 0 is used. The precision value affects the following conversions:

For **d**, **i**, **o**, **u**, **x**, and **X** (integer) conversions, the precision specifies the minimum number of digits to appear.

For **e**, **E**, and **f** (fixed-precision, floating-point) conversions, the precision specifies the number of digits to appear after the decimal-point character.

For **g** and **G** (variable-precision, floating-point) conversions, the precision specifies the maximum number of significant digits to appear.

For **s** (string) conversions, the precision specifies the maximum number of characters to appear.

A type length specifier affects the conversion as follows:

h	Causes a d, i, o, u, x, or X (integer) conversion to process a short int or unsigned short int argument; note that although the argument can have been promoted as part of the function call, the value is converted to the smaller type before it is converted.
	It causes an n (converted length assignment) operation to assign the converted length to an object of type unsigned short int.
l	Causes a d, i, o, u, x, or X (integer) conversion to process a long int or unsigned long int argument.
	It causes an n (converted length assignment) operation to assign the converted length to an object of type unsigned long int.
F	Causes the pointer associated with n, p, or s conversions to be treated as a far pointer.
L	Causes an e, E, f, g, or G (double) conversion to process a long double argument.
N	Causes the pointer associated with n, p, or s conversions to be treated as a near pointer.

The valid conversion type specifiers are:

c	An argument of type int is converted to a value of type char and the corresponding ASCII character code is written to the output stream.
d, i	An argument of type int is converted to a signed decimal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.
e, E	An argument of type double is converted to a decimal notation

e, E	An argument of type double is converted to a decimal notation in the form [-]d.ddde[+|-]ddd similar to FORTRAN exponential (**E**) notation. The leading sign appears (subject to the format control flags) only if the argument is negative. If the argument is nonzero, the digit before the decimalpoint character is nonzero. The precision is used as the number of digits following the decimalpoint character. If the precision is not specified, a default precision of 6 is used. If the precision is 0, the decimalpoint character is suppressed. The value is rounded to the appropriate number of digits. For **E** conversions, the exponent begins with the character **E** rather than **e**. The exponent sign and a three-digit number (that indicates the power of ten by which the decimal fraction is multiplied) are always produced.
f	An argument of type double is converted to a decimal notation in the form [-]ddd.ddd similar to FORTRAN fixed-point (**F**) notation. The leading sign appears (subject to the format control flags) only if the argument is negative. The precision is used as the number of digits following the decimalpoint character. If the precision is not specified, a default precision of 6 is used. If the precision is 0, the decimalpoint character is suppressed, otherwise, at least one digit is produced before the decimal-point character. The value is rounded to the appropriate number of digits.
g, G	An argument of type double is converted using either the **f** or e (or **E**, for a **G** conversion) style of conversion depending on the value of the argument. In either case, the precision specifies the number of significant digits that are contained in the result. The **e** style conversion is used only if the exponent from such a conversion would be less than -4 or greater than the precision. Trailing zeros are removed from the result and a decimal-point character only appears if it is followed by a digit.
n	The number of characters that have been written to the output stream is assigned to the integer pointed to by the argument. No output is produced.
o	An argument of type int is converted to an unsigned octal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.
p, P	An argument of type void * is converted to a value of type int and the value is formatted as for a hexadecimal (**x**) conversion.
s	Characters from the string specified by an argument of type char *, up to, but not including, the terminating NULL character (\0), are written to the output stream. If a precision is specified, no more than that many characters are written.
S	Characters from a length-preceded string are written to the output stream. If a precision is specified, no more than that many characters are written.
u	An argument of type int is converted to an unsigned decimal notation and written to the output stream. The default precision

	is 1, but if more digits are required, leading zeros are added.
x, X	An argument of type int is converted to an unsigned hexadecimal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added. Hexadecimal notation uses digits (0 through 9) and characters (a through f or A through F) for **x** or **X** conversions respectively, as the hexadecimal digits. Subject to the alternate-form control flag, **0x** or **0X** is affixed to the output.

Any other conversion type specifier character, including another percent (%) character, is written to the output stream with no special interpretation.

The arguments must correspond with the conversion type specifiers, left to right in the string; otherwise, indeterminate results occur.

For example, a specifier of the form %8.*f defines a field to be at least 8 characters wide and gets the next argument for the precision to be used in the conversion.

The output from

```
printf ("f1 = %8.4f f2 = %10.2E x = %#08x i = %d",
      23.45,   3141.5926,   0x1db,     -1 );
```

would be

```
f1 =  23.4500 f2 =   3.14E+003 x = 0x0001db i = -1
```

See Also

fprintf, **sprintf**, **vfprintf**

Example

printf

```
#include <stdio.h>

main ()
{
   char    *weekday, *month;
   int     day, year;
   weekday = "Saturday";
   month = "April";
   day = 18;
   year = 1991;
   printf ("%s, %s %d, %d\n", weekday, month, day, year);
}
```

produces the following:

```
Saturday, April 18, 1991
```

putc

Writes a character to the output stream (function or macro)
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int putc (
   int    c,
   FILE   *fp);
```

Parameters

c

(IN) Specifies the character to be written.

fp

(IN) Points to the file.

Return Values

The **putc** function or macro returns the character written. If a write error occurs, the error indicator is set and **putc** returns EOF.

Remarks

The **putc** function is equivalent to **fputc**, except that it can be implemented as a macro. The **putc** function or macro writes the character specified by the argument *c* to the output stream designated by *fp*.

See Also

ferror, **fopen**, **fputs**, **puts**

Example

putc

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

```
main ()
{
   FILE    *fp;
   int    c;
   fp = fopen ("data.fil", "r");
   while ( (c = fgetc( fp )) != EOF)
      putc (c, stdout);
   fclose (fp);
}
```

putchar

Writes a character to the output stream (function or macro)
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int putchar (
   int   c);
```

Parameters

c

(IN) Specifies the character to be written.

Return Values

This function or macro returns the character written. If a write error occurs, the error indicator is set and **putchar** returns EOF. If an error occurs, *errno* is set.

Remarks

The **putchar** function or macro writes the character specified by the argument *c* to the output stream *stdout*.

The function is equivalent to:

```
fputc (c, stdout);
```

See Also

fputc, **fputs**

Example

putchar

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

```
main ()
{
   FILE    *fp;
   int     c;
   fp = fopen ("data.fil", "r");
   c = fgetc (fp);
   while (c != EOF)
   {
      putchar (c);
      c = fgetc (fp);
   };
   fclose (fp);
}
```

puts

Writes a specified character string to the output stream and appends a newline character to the output

Local Servers: blocking

Remote Servers: blocking

Classification: ANSI

SMP Aware: No

Service: Stream I/O

Syntax

```
#include <stdio.h>

int puts (
   const char   *buf);
```

Parameters

buf

(IN) Points to the character string.

Return Values

The **puts** function returns a nonzero value if an error occurs; otherwise, it returns a value of 0. If an error occurs, *errno* is set.

Remarks

The **puts** function writes the character string pointed to by *buf* to the output stream designated by *stdout* and appends a newline character to the output. The terminating NULL character is not written.

See Also

fputs, **putc**

Example

puts

```
#include <stdio.h>

main ()
{
```

```
   FILE    *fp;
   char    buffer[80];
   fp = freopen ("data.fil", "r", stdin);
   while (gets (buffer) != NULL)
      puts (buffer);
   fclose (fp);
}
```

rewind

Sets the file position indicator to the beginning of the file
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: Yes
Service: Stream I/O

Syntax

```
#include <stdio.h>

void rewind (
   FILE   *fp);
```

Parameters

fp
(IN) Points to the file.

Return Values

None

Remarks

The **rewind** function sets the file position indicator for the stream indicated by *fp* to the beginning of the file. It is equivalent to:

```
fseek( fp, 0L, SEEK_SET );
```

except that the error indicator for the stream is cleared.

See Also

clearerr, **fopen**

Example

rewind

```
#include <stdio.h>

FILE    *fp;
```

```
if ( (fp = fopen ("program.asm", "r") ) != NULL)
{
   assemble_pass ( 1 );
   rewind (fp);
   assemble_pass ( 2 );
   fclose (fp);
}
```

scanf

Scans input from a stream
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: Yes
Service: Stream I/O

Syntax

```
#include <stdio.h>

int scanf (
   const char   *format,
   ... );
```

Parameters

format

(IN) Points to the format control string.

Return Values

The **scanf** function returns EOF when the scanning is terminated by reaching the end of the input stream. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

Remarks

The **scanf** function scans input from the file designated by *stdin* under control of the argument *format*. Following the format string is the list of addresses of items to receive values.

Format Control String

The format control string consists of zero or more format directives that specify acceptable input file data. Subsequent arguments are pointers to various types of objects that are assigned values as the format string is processed.

A format directive can be a sequence of one or more white-space characters, an ordinary character, or a conversion specifier. An ordinary character in the format string is any character, other than a white-space character or the percent (%) character, that is not part of a conversion specifier. A conversion specifier is a sequence of characters in the format

string, which begins with a % and is followed, in sequence, by the following:

An optional assignment suppression indicator: the asterisk character (*)

An optional decimal integer that specifies the maximum field width to be scanned for the conversion

An optional pointer type specification: one of **N** or **F**

An optional type length specification: one of **h**, **l** or **L**

A character that specifies the type of conversion to be performed. It can be any one of the following characters:

c	d	e	f	g	i	o	n	p	s	u	x

As each format directive in the format string is processed, the directive can successfully complete, fail because of a lack of input data, or fail because of a matching error as defined by the particular directive. If end-of-file is encountered on the input data before any characters that match the current directive have been processed (other than leading white-space where permitted), the directive fails for lack of data. If end-of-file occurs after a matching character has been processed, the directive is completed (unless a matching error occurs), and the function returns without processing the next directive. If a directive fails because of an input character mismatch, the character is left unread in the input stream. Trailing white-space characters, including newline characters, are not read unless matched by a directive. When a format directive fails, or the end of the format string is encountered, the scanning is completed and the function returns.

When one or more white-space characters---space, horizontal tab (\t), vertical tab (\v), form feed (\f), carriage return (\r), newline or line feed (\n)---occur in the format string, input data up to the first nonwhite-space character is read, or until no more data remains. If no white-space characters are found in the input data, the scanning is complete and the function returns.

An ordinary character in the format string is expected to match the same character in the input stream.

A conversion specifier in the format string is processed as follows:

For conversion types other than [, **c**, and **n**, leading white-space characters are skipped.

For conversion types other than n, all input characters, up to any specified maximum field length, that can be matched by the conversion type are read and converted to the appropriate type of value; the character immediately following the last character to be

matched is left unread; if no characters are matched, the format directive fails.

Unless the assignment suppression indicator (*) was specified, the result of the conversion is assigned to the object pointed to by the next unused argument (if assignment suppression was specified, no argument is skipped); the arguments must correspond in number, type, and order to the conversion specifiers in the format string.

A pointer type specification is used to indicate the type of pointer used to locate the next argument to be scanned:

F	Points to a far pointer.
N	Points to a near pointer.

The pointer type defaults to that used for data in the memory model for which the program has been compiled.

A type length specifier affects the conversion as follows:

h	Causes a **d**, **i**, **o**, **u**, or **x** (integer) conversion to assign the converted value to an object of type short int or unsigned short int. It causes an **n** (read length assignment) operation to assign the number of characters that have been read to an object of type unsigned short int.
l	Causes a **d**, **i**, **o**, **u**, or **x** (integer) conversion to assign the converted value to an object of type long int or unsigned long int. It causes an **n** (read length assignment) operation to assign the number of characters that have been read to an object of type unsigned long int. It causes an **e**, **f**, or **g** (floatingpoint) conversion to assign the converted value to an object of type double.
L	Causes an **e**, **f**, or **g** (floatingpoint) conversion to assign the converted value to an object of type long double.

The valid conversion type specifiers are:

c	Any sequence of characters in the input stream of the length specified by the field width, or a single character if no field width is specified, is matched. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence, without a terminating NULL character (\0). For a single-character assignment, a pointer to a single object of type char is sufficient.

d	A decimal integer, consisting of an optional sign, followed by one or more decimal digits, is matched. The argument is assumed to point to an object of type int.
e, **f**, **g**	A floating-point number, consisting of an optional sign (+ or -), followed by one or more decimal digits, optionally containing a decimal-point character, followed by an optional exponent of the form **e** or **E**, an optional sign, and one or more decimal digits, is matched. The exponent, if present, specifies the power of ten by which the decimal fraction is multiplied. The argument is assumed to point to an object of type float.
i	An optional sign, followed by an octal, decimal, or hexadecimal constant is matched. An octal constant consists of zero and zero or more octal digits. A decimal constant consists of a nonzero decimal digit and zero or more decimal digits. A hexadecimal constant consists of the characters **0x** or 0X followed by one or more (upper- or lowercase) hexadecimal digits. The argument is assumed to point to an object of type int.
n	No input data is processed. Instead, the number of characters that have already been read is assigned to the object of type unsigned int that is pointed to by the argument. The number of items that have been scanned and assigned (the return value) is not affected by the n conversion type specifier.
o	An octal integer, consisting of an optional sign, followed by one or more (zero or nonzero) octal digits, is matched. The argument is assumed to point to an object of type int.
p	A hexadecimal integer, as described for **x** conversions below, is matched. The converted value is further converted to a value of type void* and then assigned to the object pointed to by the argument.
s	A sequence of nonwhite-space characters is matched. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence and a terminating NULL character, which is added by the conversion operation.
u	An unsigned decimal integer, consisting of one or more decimal digits, is matched. The argument is assumed to point to an object of type unsigned int.
x	A hexadecimal integer, consisting of an optional sign, followed by an optional prefix 0x or 0X, followed by one or more (uppercase or lowercase) hexadecimal digits, is matched. The argument is assumed to point to an object of type int.
[**c1c2** ...]	A sequence of characters, consisting of any of the characters c1, **c2**, ... called the scanset, in any order, is matched. **c1** cannot be the caret character (^). If **c1** is], that character is considered to be part of the scanset and a second] is required to end the format directive. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence and a terminating NULL character, which is added

	sequence and a terminating NULL character, which is added by the conversion operation.
[^**c1c2** ...]	A sequence of characters, consisting of any of the characters other than the characters between the ^ and], is matched. As with the preceding conversion, if **c1** is], it is considered to be part of the scanset and a second] ends the format directive. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence and a terminating NULL character, which is added by the conversion operation.

A conversion type specifier of % is treated as a single ordinary character that matches a single % character in the input data. A conversion type specifier other than those listed above causes scanning to terminate the function to return.

The line

```
scanf ("%s%*f%3hx%d", name, &hexnum, &decnum)
```

with input

```
some_string 34.555e-3 abc1234
```

copies some_string into the array name, skip 34.555e-3, assign 0xabc to hexnum and 1234 to decnum. The return value is 3.

The line

```
char fmt[100];
   strcpy (fmt, "%[abcdefghijklmnopqrstuvwxyz");
   strcat (fmt,"[ABCDEFGHIJKLMNOPQRSTUVWZ]%*2s%[W\n]");
   scanf (fmt, string1, string2)
```

with input

```
They may look alike, but they don't perform alike.
```

assigns

```
"They may look alike"
```

to string1, skip the comma and the space, and assign

```
" but they don't perform alike.".
```

to string2. (The %*2s only matches the ","; the next blank terminates that field.)

See Also

fscanf, **sscanf**

Example

scanf

To scan a date in the form "Saturday April 18 1991":

```
#include  <stdio.h>
int day, year;
char weekday[10], month[12];
scanf ("%s %s %d %d", weekday, month, &day, &year);
```

setbuf

Associates a buffer with a file after the file is open and before it has been read or written to

Local Servers: nonblocking

Remote Servers: nonblocking

Classification: ANSI

SMP Aware: Yes

Service: Stream I/O

Syntax

```
#include <stdio.h>

void setbuf (
   FILE   *fp,
   char   *buffer);
```

Parameters

fp

(IN) Points to the file.

buffer

(IN) Points to the buffer.

Return Values

The **setbuf** function returns no value.

Remarks

The **setbuf** function can be used to associate a buffer with the file designated by *fp*. If this function is used, it must be called after the file has been opened and before it has been read or written. If the argument buffer is NULL, then all input/ output for the file pointed to by*fp* is completely unbuffered. If the argument buffer is not NULL, then it must point to an array that is at least BUFSIZ characters in length, and all input/output is fully buffered. BUFSIZ is a constant defined in STDIO.H.

See Also

fopen, **setvbuf**

Example

setbuf

```
#include <stdio.h>

main ()
{
   char    *buffer;
   FILE    *fp;
   fp = fopen ("data.fil", "r");
   buffer = malloc (BUFSIZ);
   setbuf (fp, buffer);
   fclose(fp);
}
```

setvbuf

Associates a buffer with a file after the file is open and before it has been read or written to

Local Servers: nonblocking

Remote Servers: nonblocking

Classification: ANSI

SMP Aware: No

Service: Stream I/O

Syntax

```
#include <stdio.h>

int setvbuf (
   FILE      *fp,
   char      *buf,
   int       mode,
   size_t    size);
```

Parameters

fp

(IN) Points to the file.

buf

(IN) Points to the buffer.

mode

(IN) Specifies the file mode that determines how to buffer the file.

size

(IN) Specifies the size of the array.

Return Values

The **setvbuf** function returns a value of 0 on success, or a nonzero value if an invalid value is given for *mode* or *size*.

Remarks

The **setvbuf** function can be used to associate a buffer with the file designated by *fp*. If this function is used, it must be called after the file has been opened and before it has been read or written. The argument *mode* determines how the file pointed to by *fp* is to be buffered, as follows:

_IOFBF	Causes input/output to be fully buffered.
_IOLBF	Causes output to be line buffered (the buffer is flushed when a newline character is written, when the buffer is full, or when input is requested).
_IONBF	Causes input/output to be completely unbuffered.

If the argument *buf* is not NULL, the array to which it points is used instead of an automatically allocated buffer. The argument *size* specifies the size of the array.

See Also

fopen, **setbuf**

Example

setvbuf

```
#include <stdio.h>

main ()
{
   char    *buf;
   FILE    *fp;
   fp = fopen ("data.fil", "r");
   buf = malloc (1024);
   setvbuf (fp, buf, _IOFBF, 1024);
   fclose(fp);
}
```

tmpfile

Creates a temporary binary file
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: Yes
Service: Stream I/O

Syntax

```
#include <stdio.h>

FILE *tmpfile (void);
```

Return Values

The **tmpfile** function returns a pointer to the stream of the file that it created. If the file cannot be created, the **tmpfile** function returns NULL. If an error occurs, *errno* is set.

Remarks

The **tmpfile** function creates a temporary binary file that is automatically removed when it is closed or at program termination. The file is opened for update.

See Also

fopen, **freopen**, **tmpnam**

Example

tmpfile

```
#include <stdio.h>

main ()
{
   static FILE    *TempFile;
   TempFile = tmpfile ();
}
```

ungetc

Pushes a character back onto the specified input stream
Local Servers: nonblocking
Remote Servers: nonblocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

```
#include <stdio.h>

int ungetc (
   int    c,
   FILE   *fp);
```

Parameters

c

(IN) Specifies the character to be pushed back onto the specified input stream.

fp

(IN) Specifies the input stream.

Return Values

The **ungetc** function returns the character pushed back.

Remarks

The **ungetc** function pushes the character specified by *c* back onto the input stream specified by *fp*. This character is returned by the next read from the stream. Only the last character returned in this way is remembered.

The **ungetc** function clears the EOF indicator, unless the value of *c* is EOF.

See Also

fopengetc

vfprintf

Writes output to a stream under format control
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: No
Service: Stream I/O

Syntax

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

int vfprintf (
   FILE         *fp,
   const char   *format,
   va_list      arg);
```

Parameters

fp

(IN) Points to to the file.

format

(IN) Points to to the format control string.

arg

(IN) Specifies a variable argument.

Return Values

The **vfprintf** function returns the number of characters written or a negative value if an output error occurred. If an error occurs,*errno* is set.

Remarks

The **vfprintf** function writes output to the file pointed to by*fp* under control of the argument *format*. The format string is described under the description for **printf**. The **vfprintf** function is equivalent to **fprintf**, with the variable argument list replaced with*arg*, which has been initialized by the **va_start** macro.

See Also

fprintf, **printf**, **sprintf**, **va_arg**, **va_end**, **va_start**

Example

vfprintf

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

extern FILE *LogFile;

void errmsg          /* A GENERAL ERROR ROUTINE */
(char *format, ... )
{
   va_list arglist;
   va_start (arglist, format);
   vfprintf (stderr, format, arglist);
   va_end  (arglist );
   if  (LogFile != NULL)
   {
      va_start (arglist, format);
      vfprintf (LogFile, format, arglist);
      va_end (arglist);
   }
}
```

vfscanf

Scans input from a stream under format control
Local Servers: blocking
Remote Servers: blocking
Classification: Other
SMP Aware: No
Service: Stream I/O

Syntax

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

int vfscanf (
   FILE         *fp,
   const char   *format,
   va_list      arg);
```

Parameters

fp

(IN) Points to the file.

format

(IN) Points to the format control string.

arg

(OUT) Specifies a variable argument.

Return Values

The **vfscanf** function returns EOF when the scanning is terminated by reaching the end of the input stream. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned. If a file input error occurs, *errno* is set.

Remarks

The **vfscanf** function scans input from the file designated by *fp* under control of the argument *format*. The format list is described with the **scanf** function.

The **vfscanf** function is equivalent to the **fscanf** function, with a variable argument list replaced with *arg*, which has been initialized using the **va_start** macro.

See Also

fscanf, scanf, sscanf, va_arg, va_end, va_start

Example

vfscanf

```
#include <stdio.h>

#include <stdarg.h>
main ()
{
   auto va_list arglist;
   va_start (arglist, arg);
   vfprintf (fp, format, arglist);
   va_end (arglist);
}
```

vprintf

Writes output to *stdout* under format control
Local Servers: blocking
Remote Servers: blocking
Classification: ANSI
SMP Aware: Yes
Service: Stream I/O

Syntax

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

int vprintf (
   const char   *format,
   va_list      arg);
```

Parameters

format

(IN) Points to to the format control string.

arg

(IN) Specifies a variable argument.

Return Values

The **vprintf** function returns the number of characters written or a negative value if an output error occurred. If an error occurs, *errno* is set.

Remarks

The **vprintf** function writes output to the file *stdout* under control of the argument *format*. The format string is described under the description for **printf**. The **vprintf** function is equivalent to **printf**, with the variable argument list replaced with *arg*, which has been initialized by the **va_start** macro.

See Also

fprintf, **printf**, **sprintf**, **va_arg**, **va_end**, **va_start**

Example

vprintf

vprintf

The following shows the use of **vprintf** in a general error message routine.

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

void errmsg (char *format, ... )
{
   va_list arglist;
   printf ("Error: ");
   va_start (arglist, format);
   vprintf (format, arglist);
   va_end (arglist);
}
```

vscanf

Scans input from the stream designated by *stdin*
Local Servers: blocking
Remote Servers: blocking
Classification: Other
SMP Aware: Yes
Service: Stream I/O

Syntax

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

int vscanf (
   const char   *format,
   va_list      arg);
```

Parameters

format

(IN) Points to to the format control string.

arg

(OUT) Specifies a variable argument.

Return Values

The **vscanf** function returns EOF when the scanning is terminated by reaching the end of the input stream. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

Remarks

The **vscanf** function scans input from the file designated by *stdin* under control of the argument *format*. The format list is described with the **scanf** function.

The **vscanf** function is equivalent to the **scanf** function, with a variable argument list replaced with *arg*, which has been initialized using the **va_start** macro

See Also

fscanf, **scanf**, **sscanf**, **va_arg**, **va_end**, **va_start**

Example

vscanf

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

void find (char *format, char *arg, ... )
{
   va_list arglist;
   va_start (arglist, arg);
   vscanf (format, arglist);
   va_end (arglist );
}
```

Synchronization

Synchronization: Guides

Synchronization: Task Guide

Synchronization: Concept Guide

Semaphore Functions

Additional Links

Synchronization: Functions

Synchronization: Structures

Parent Topic:

Synchronization: Guides

Types of Data Locks

NetWare® supports three types of data locks:

File Locks

Physical Record Locks

Logical Record Locks

Parent Topic:

Synchronization: Guides

Synchronization: Tasks

Clearing Logged Files

Files remain in the file log table until you clear them.

NWClearFileLock2 clears a single file from the log table.

NWClearFileLockSet clears all files from the log table.

If a file is locked when you ask the server to clear it from the table, the server releases the lock and clears the file. The server also closes the file if it's open.

Parent Topic:

Synchronization: Guides

Related Topics:

Logging Files

Releasing Locked Files

Locking Data and Files

This section outlines steps for locking data. Locking procedures are built around the file log table. The NetWare® server maintains a log table for each connection task. (In multi-tasking environments, one task's log table is not affected by another's.)

To lock one or more files, log the files into the table and then request a file lock. If the server can't lock all the files, the operation fails and none of the files are locked. This method protects your application from entering deadlock with another application, a situation in which each application is waiting for the other to release a partially locked set of files.

The following steps explain file locking, but the basic steps are the same for locking files, physical records, or logical records.

1. **Call NWLogFileLock2 for each file you intend to lock. This will log each file into the log file. The *timeOut* parameter controls the duration of the server's efforts.**

2. **After you have logged all files that you intend to lock, call NWLockFileLockSet. This will lock all the files at once. Again, a**

timeout value controls the duration of the server's efforts.

3. **Use the files you have locked.**
4. **Release the locks on the files you have locked. To release individual locks, call NWReleaseFileLock2 for each file. To release the entire set of files, call NWReleaseFileLockSet.**
5. **Clear the files from the file log table. To clear individual files, call NWClearFileLock2 for each file. To clear the entire set of files, call NWClearFileLockSet.**

   NOTE: If a file is locked when you ask the server to clear it from the log table, the server releases the lock and clears the file. The server also closes the file if it's open.

Logging Files

Locking Files

Releasing Locked Files

Clearing Logged Files

Parent Topic:

Synchronization: Guides

Locking Files

After you have logged all the files you intend to lock, call **NWLockFileLockSet**. This function locks all the files at once. The value of the *timeOut* parameter controls the duration of the server's efforts.

To lock one or more files, log the files into the table and then request a file lock. If the server can't lock all the files, the operation fails and none of the files are locked. This method protects your application from entering deadlock with another application, a situation in which each application is waiting for the other to release a partially locked set of files.

Parent Topic:

Synchronization: Guides

Related Topics:

Logging Files

Logging Files

Locking procedures are built around the file log table. The NetWare® server

maintains a log table for each connection task. (In multi-tasking environments, one task's log table is not affected by another's.)

Call **NWLogFileLock2** to log files in the log table. Call this function for each file you intend to lock. The *timeOutLimit* parameter lets you control the amount of time the server spends attempting to lock the file. You can also have the server enter the file into the table without attempting to lock it.

Parent Topic:

Synchronization: Guides

Related Topics:

Locking Files

Releasing Locked Files

Files remain locked until you specifically ask the server to release them. You can release a file lock on an individual file or on an entire set of locked files:

NWReleaseFileLock2 releases a lock on a single file.

NWReleaseFileLockSet releases the lock on a set of files.

Parent Topic:

Synchronization: Guides

Related Topics:

Locking Files

Clearing Logged Files

Synchronization: Concepts

File Locking Functions

These functions manage file locks.

Function	Comment
NWClearFileLock2	Unlocks the specified file and removes it from the log table.
NWClearFileLockSet	Unlocks and removes all files logged in the log table.
NWLockFileLockSet	Locks all files logged in the log table.
NWLogFileLock2	Logs the specified file in the log table.
NWReleaseFileLock2	Unlocks the specified file but doesn't remove it from the log table.
NWReleaseFileLockSet	Unlocks all logged files but doesn't remove them from the log table.

Parent Topic:

Synchronization: Guides

File Locks

File locks control access to an entire file or several files at the same time. Once locked, a file can't be accessed by another connection.

Parent Topic:

Types of Data Locks

Related Topics:

Physical Record Locks

Logical Record Locks

Logical Record Locking Functions

These functions manage logical record locks.

Function	Comment
NWClearLogicalRecord	Unlocks the specified logical record and removes it from the log table.
NWClearLogicalRecordSet	Unlocks and then removes all logical records logged in the log table.
NWLogLogicalRecord	Logs the specified logical record in the log table.
NWLockLogicalRecordSet	Locks all logical records logged in the log table.
NWReleaseLogicalRecord	Unlocks the specified logical record but doesn't remove it from the log table.
NWReleaseLogicalRecordSet	Unlocks all log logical records but doesn't remove them from the log table.

Parent Topic:

Synchronization: Guides

Logical Record Locks

Logical record locks control access to a logical record name. You define logical record names for the purposes of your application. By associating the logical record with a specific file or physical record you can coordinate access to data within your application.

The NetWare® server only reports the status of the logical record. It's up to your application to enforce restrictions implied by a logical record. The server doesn't prevent applications from accessing physical data you have associated with the logical record. For this reason, we recommend you don't rely on file or physical record locks when using logical record locks.

Parent Topic:

Types of Data Locks

Related Topics:

File Locks

Physical Record Locks

Physical Record Locking Functions

These functions manage physical record locks.

Function	Comment
NWClearPhysicalRecord	Unlocks the specified physical record and removes it from the log table.
NWClearPhysicalRecordSet	Unlocks all logged physical records and removes them from the log table.
NWLockPhysicalRecordSet	Unlocks and removes all physical records from the log table.
NWLogPhysicalRecord	Logs a physical record in the log table.
NWReleasePhysicalRecord	Unlocks the specified physical record but doesn't remove it from the log table.
NWReleasePhysicalRecordSet	Unlocks all logged physical records but doesn't remove them from the log table.

Parent Topic:

Synchronization: Guides

Physical Record Locks

Physical record locks control access to byte ranges within a file. To lock a physical record, specify a starting offset within the file and the length of the record in bytes. Only the byte range is locked; the rest of the file remains free. Physical record locks can be exclusive or shareable.

Parent Topic:

Types of Data Locks

Related Topics:

Logical Record Locks

File Locks

Semaphores

Like a logical record lock, a semaphore is used to control access to data stored on the NetWare® server. As with logical records, you are responsible for defining and enforcing any restrictions associated with a semaphore. A common use for semaphores is to limit the number of users who can access a network application.

Unlike a logical record, a semaphore allows you to configure the number of applications that can access the data. You can scan for the semaphores that a connection has open, as well as scan for the connections that have opened a semaphore.

To set up a semaphore, call **NWOpenSemaphore**. This function takes the name of the semaphore and an initial semaphore value. The initial value is zero based and indicates the number of applications that can access the semaphore. For example, if the initial value is 4, five applications can access the semaphore (one of which is the application that opened the semaphore).

After the semaphore is open, applications needing access to the resource associated with the semaphore must call **NWWaitOnSemaphore**. This function decrements the semaphore. If the resulting value is zero or greater, the function returns successfully. In that case, the resource is considered available. If the semaphore reaches a negative value, the application must wait until the semaphore returns to zero before accessing the resource.

When an application finishes using the protected resource, it calls **NWCloseSemaphore**. This function decrements the semaphore's open count by one. The semaphore is deleted by the last process to call this function.

Parent Topic:

Synchronization: Guides

Semaphore Functions

These functions manage semaphores.

Function	Comment
NWCloseSemaphore	Closes a semaphore and decrements the open count.
NWExamineSemaphore	Returns the semaphore value and the number of workstations that have the semaphore open.
NWOpenSemaphore	Creates and initializes a named semaphore to the specified value.
NWSignalSemaphore	Increments the semaphore value by one.
NWWaitOnSemaphore	Allows the application to queue up for access to the resource associated with a semaphore.

Parent Topic:

Synchronization: Guides

Synchronization Introduction

Synchronization provides developers with the ability to lock users out of a file while it is being accessed by someone else. Synchronization is essential to assuring data integrity on the network, where many users can access the same data simultaneously. Data locks are the basis for controlling file access.

NetWare® also supports semaphores for controlling access to files. However, semaphores can be applied to other resources as well, and so aren't exclusively a file synchronization mechanism.

In addition to locking data, you can scan for information about locks and semaphores such as a list of locks associated with a specified connection or a list of connections locking a specified file.

> **NOTE:** NetWare 3.11 introduced numerous new NCP requests for file locking and semaphore management. Synchronization takes advantage of the new requests whenever possible.

Parent Topic:

Synchronization: Guides

Synchronization Scan Functions

These functions scan for synchronization information in association with workstation connections.

Function	Comment
NWScanLogicalLocksByConn	Scans for all logical record locks on a specified connection.
NWScanLogicalLocksByName	Scans for all record locks on a specified logical name.
NWScanPhysicalLocksByConnFile	Scans for all physical record locks on a specified connection for a specified file.
NWScanPhysicalLocksByFile	Scans for all record locks on a specific physical file.
NWScanSemaphoresByConn	Scans information about the semaphores that a specified connection has open.
NWScanSemaphoresByName	Scans information about a semaphore by name.

Parent Topic:

Synchronization: Guides

Synchronization: Functions

NWClearFileLock2

Unlocks the specified file and removes it from the log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWClearFileLock2 (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWClearFileLock2
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle of the directory containing the locked file.

path

(IN) Points to the string containing the name and path of the locked file.

Return Values

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

G0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89A1	DIRECTORY_IO_ERROR
0x89FD	BAD_STATION_NUMBER
0x89FF	LOCK_ERROR

Remarks

To avoid deadlock, a workstation must request those resources it needs to lock by making an entry in the File Log Table at the NetWare server. Once the log table is complete, the application attempts to lock those records. Locking works only if all records in the table are available. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

2.x servers will also return INVALID_PATH when a bad directory handle is passed.

If the file is open, **NWClearFileLock2** causes it to be closed on the server. The application should close the associated file on the workstation to clear the local file handle correctly.

path can specify either a file's complete path name or a path relative to the current working directory. For example, if a file's complete path name is SYS:ACCOUNT/DOMEST/TARGET.DAT and the directory handle mapping is SYS:ACCOUNT, *path* could point to either of the following:

```
SYS:ACCOUNT/DOMEST/TARGET.DAT
DOMEST/TARGET.DAT
```

NCP Calls

0x2222 07 Clear File

See Also

NWClearFileLockSet, **NWLogPhysicalRecord**, **NWLogFileLock2**

NWClearFileLockSet

Unlocks all files logged in the File Log Table and removes them from the log table

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWClearFileLockSet (
   void);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWClearFileLockSet
  : NWCCODE;
```

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

To avoid deadlock, a workstation must request those resources it needs to lock by making an entry in the File Log Table at the NetWare server. Once the log table is complete, the application attempts to lock those records. Locking works only if all records in the table are available. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

All open files in the task's log table are closed. The file handles on the workstation itself are not cleared---this should be done by the application

and any error codes should be ignored. **NWClearFileLockSet** is ignored if the associated task on the workstation does not have logged files.

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

NWClearFileLock2, **NWLogFileLock2**, **NWReleaseFileLock2**, **NWReleaseFileLockSet**

NWClearLogicalRecord

Unlocks a logical record and removes it from the log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWClearLogicalRecord (
   NWCONN_HANDLE   conn,
   pnstr8          logRecName);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWClearLogicalRecord
  (conn : NWCONN_HANDLE;
   logRecName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle containing the logical record.

logRecName

(IN) Points to the name of the logical record being cleared (128 characters).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

0x89FF	LOCK_ERROR

Remarks

A logical record is simply a name (a string) registered with the NetWare server. The name (as with a semaphore) can then be locked or unlocked by applications and can be used as an inter-application locking mechanism.

NOTE: Locking or unlocking a logical record does not physically lock or unlock those resources associated with the logical record; only the applications using the record know about such an association.

Applications define logical record names. A logical record name represents a group of files, physical records, structures, etc. **NWLogLogicalRecord** or **NWLockLogicalRecordSet** lock one or more logical record names, not the actual files, physical records, or structures associated with each logical record name. Any uncooperative application can ignore a lock on the logical record name and directly lock physical files or records. Therefore, applications using logical record locks must not use other locking techniques simultaneously.

NCP Calls

0x2222 11 Clear Logical Record

See Also

NWClearLogicalRecordSet, **NWLockLogicalRecordSet**, **NWLogLogicalRecord**, **NWReleaseLogicalRecord**, **NWReleaseLogicalRecordSet**

NWClearLogicalRecordSet

Unlocks and then removes all of the logical records logged in the log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWClearLogicalRecordSet (
   void);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWClearLogicalRecordSet
  : NWCCODE;
```

Return Values

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

0x0000	SUCCESSFUL

Remarks

A logical record is simply a name (a string) registered with the NetWare server. The name (as with a semaphore) can then be locked or unlocked by applications and can be used as an inter-application locking mechanism.

NOTE: Locking or unlocking a logical record does not physically lock or unlock those resources associated with the logical record; only the applications using the record know about such an association.

If the requesting process does not have logged logical records, **NWClearLogicalRecordSet** is ignored.

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

NWClearLogicalRecord, **NWLockLogicalRecordSet**, **NWLogLogicalRecord**, **NWReleaseLogicalRecord**, **NWReleaseLogicalRecordSet**

NWClearPhysicalRecord

Unlocks the specified physical record and removes it from the log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWClearPhysicalRecord (
   NWFILE_HANDLE   fileHandle,
   nuint32         recStartOffset,
   nuint32         recSize);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWClearPhysicalRecord
  (fileHandle : NWFILE_HANDLE;
   recStartOffset : nuint32;
   recSize : nuint32
) : NWCCODE;
```

Parameters

fileHandle

(IN) Specifies the file handle associated with the file containing the physical record being cleared.

recStartOffset

(IN) Specifies the offset, from the beginning of the file, at which the record starts.

recSize

(IN) Specifies the length, in bytes, of the locked record.

Return Values

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

0x0000	SUCCESSFUL
0x8988	INVALID_FILE_HANDLE
0x89FF	LOCK_ERROR

Remarks

NWClearPhysicalRecord locates the physical record within the specified file by passing the offset in *recStartOffset* and the length in *recSize*.

NOTE: Locking or unlocking a logical record does not physically lock or unlock those resources associated with the logical record; only the applications using the record know about such an association.

recStartOffset and *recSize* should match the corresponding parameters in **NWLogPhysicalRecord**.

NWClearPhysicalRecord is ignored if the requesting workstation does not have logged physical records.

NCP Calls

0x2222 30 Sync Clear Physical Record

See Also

NWClearPhysicalRecordSet, **NWLockPhysicalRecordSet**, **NWLogPhysicalRecord**, **NWReleasePhysicalRecord**, **NWReleasePhysicalRecordSet**

NWClearPhysicalRecordSet

Unlocks and removes all physical records from the log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWClearPhysicalRecordSet (
   void);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWClearPhysicalRecordSet
  : NWCCODE;
```

Return Values

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

0x0000	SUCCESSFUL

Remarks

NWClearPhysicalRecordSet is ignored if the requesting workstation does not have logged or locked physical records.

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

NWClearPhysicalRecord, **NWLockPhysicalRecordSet**, **NWLogPhysicalRecord**, **NWReleasePhysicalRecord**, **NWReleasePhysicalRecordSet**

NWCloseSemaphore

Closes a semaphore and decrements the open count of the semaphore, indicating one less process is holding the semaphore open

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWCloseSemaphore (
   NWCONN_HANDLE   conn,
   nuint32         semHandle);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWCloseSemaphore
  (conn : NWCONN_HANDLE;
   semHandle : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

semHandle

(IN) Specifies the semaphore handle obtained when the semaphore was opened by **NWOpenSemaphore**.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

0x89FF	INVALID_SEMAPHORE_HANDLE, LOCK_ERROR

Remarks

If the requesting process is the last process to have this semaphore open, the semaphore is deleted.

NCP Calls

0x2222 32 4 Close Semaphore

See Also

NWExamineSemaphore, **NWOpenSemaphore**, **NWSignalSemaphore**, **NWWaitOnSemaphore**

NWExamineSemaphore

Returns the semaphore value
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWExamineSemaphore (
   NWCONN_HANDLE   conn,
   nuint32         semHandle,
   pnint16         semValue,
   pnuint16        semOpenCount);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWExamineSemaphore
  (conn : NWCONN_HANDLE;
   semHandle : nuint32;
   semValue : pnint16;
   semOpenCount : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

semHandle

(IN) Specifies the semaphore handle obtained when the semaphore was opened by **NWOpenSemaphore**.

semValue

(OUT) Points to the current semaphore value (optional).

semOpenCount

(OUT) Points to the number of stations that currently have this semaphore open.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x89FF	LOCK_ERROR

Remarks

A semaphore value greater than 0 indicates the application can access the associated network resource. A negative value indicates the number of processes waiting to use the semaphore. If the semaphore value is negative, the application must either enter a waiting queue by calling **NWWaitOnSemaphore** or temporarily abandon its attempt to access the network resource.

semOpenCount indicates the number of processes holding the semaphore open. **NWOpenSemaphore** increments this value. **NWCloseSemaphore** decrements this value.

semValue is optional. Use NULL if a return value is not desired.

NCP Calls

0x2222 32 1 Examine Semaphore

See Also

NWCloseSemaphore, **NWOpenSemaphore**, **NWSignalSemaphore**, **NWWaitOnSemaphore**

NWLockFileLockSet

Locks files that have been logged by a workstation task in the File Log Table of a NetWare server

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWLockFileLockSet (
   nuint16   timeOut);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWLockFileLockSet
  (timeOut : nuint16
) : NWCCODE;
```

Parameters

timeOut

(IN) Specifies the length of time the NetWare server attempts to lock the record set before timing out.

Return Values

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

0x0000	SUCCESSFUL
0x89FE	TIMEOUT_FAILURE

Remarks

To help avoid deadlock, a workstation task can log file locks in the File

To help avoid deadlock, a workstation task can log file locks in the File Log Table of a NetWare server. When the files in the log table are needed, **NWLockFileLockSet** can be called.

NWLockFileLockSet will attempt to lock the logged set on all attached servers. Locks will be attempted by ordering the servers according to their net node addresses and making the request on each server. If the request fails at any point, **NWLockFileLockSet** will automatically release all locks made to that point.

All files on all servers must be available for **NWLockFileLockSet** to complete successfully.

There is no way to determine which server the lock request failed on.

timeOut is the length of time the NetWare server will attempt the operation before failing. This limit is specified in units of 1/18 second (0 = no wait).

In DOS and Windows, all access to the network is blocked during any time out period. For this reason, time outs should be kept to an absolute minimum---a value of 18 or less. (Even though DOS is mono-tasking, the application may be running in a DOS box under Windows Enhanced Mode.)

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

NWClearFileLock2, **NWClearFileLockSet**, **NWLogFileLock2**, **NWReleaseFileLock2**, **NWReleaseFileLockSet**

NWLockLogicalRecordSet

Locks all logical records logged in the log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWLockLogicalRecordSet (
   nuint8    lockFlags,
   nuint16   timeOut);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWLockLogicalRecordSet
  (lockFlags : nuint8;
   timeOut : nuint16
) : NWCCODE;
```

Parameters

lockFlags

(IN) Specifies the lock flags.

timeOut

(IN) Specifies the length of time the NetWare server attempts to lock the record set before timing out.

Return Values

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

0x0000	SUCCESSFUL
0x89FE	TIMEOUT_FAILURE

Remarks

Applications define logical record names. A logical record name represents a group of files, physical records, or data structures. **NWLogLogicalRecord** and **NWLockLogicalRecordSet** affect one or more logical record names, not the actual files, physical records, or data structures associated with each logical record name. Any uncooperative application can ignore a lock on the logical record name and directly lock physical files or records. Therefore, applications using logical record locks must not simultaneously use other locking techniques.

To avoid deadlock, request the resources needed to lock by making an entry in the File Log Table at the NetWare server. Once the log table is complete, the application attempts to lock those records. The locking works only if all records in the table are available. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

A logical record is simply a name (a string) registered with the NetWare server. The name (as with a semaphore) can then be locked or unlocked by applications and can be used as an inter-application locking mechanism.

NOTE: Locking or unlocking a logical record does not physically lock or unlock those resources associated with the logical record; only the applications using the record know about such an association.

lockFlags is interpreted as follows:

```
0x00   Lock record with a shareable lock
0x01   Lock record with an exclusive loc
```

timeOut is specified in units of 1/18 second (0 = no wait).

In DOS and Windows, all access to the network is blocked during any timeout period. For this reason, timeouts should be kept to an absolute minimum---a value of 18 or less. (Even though DOS is mono-tasking, the application may be running in a DOS box under Windows Enhanced Mode.)

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

NWClearLogicalRecord, **NWClearLogicalRecordSet**, **NWLogLogicalRecord**, **NWReleaseLogicalRecord**,

NWReleaseLogicalRecordSet

NWLockPhysicalRecordSet

Locks all records logged in the physical record log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWLockPhysicalRecordSet (
   nuint8    lockFlags,
   nuint16   timeOut);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWLockPhysicalRecordSet
  (lockFlags : nuint8;
   timeOut : nuint16
) : NWCCODE;
```

Parameters

lockFlags

(IN) Specifies the lock flags.

timeOut

(IN) Specifies the length of time the NetWare server attempts to lock the record set before timing out.

Return Values

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

0x0000	SUCCESSFUL
0x89FE	TIMEOUT_FAILURE

Remarks

A physical record lock, as opposed to a logical lock, is the actual lock of a specified record relative to a physical file. Before a record is locked, it is also entered in the File Log Table at the NetWare server. Records can be locked only if all records in the log table are available for locking. This avoids deadlock.

To avoid deadlock, request those resources needing to be locked by making an entry in the File Log Table at the NetWare server. Once the log table is complete, **NWLockPhysicalRecordSet** attempts to lock those records. The locking only works if all records in the table are available. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

timeOut is specified in units of 1/18 second (0 = no wait).

In DOS and Windows, all access to the network is blocked during any timeout period. For this reason, timeouts should be kept to an absolute minimum---a value of 18 or less. (Even though DOS is mono-tasking, the application may be running in a DOS box under Windows Enhanced Mode.)

lockFlags is interpreted as follows:

```
0x00   Lock records with exclusive lock
0x02   Lock records with shareable lock
```

A shareable lock prevents any process, including the one which made the lock, from writing to the record.

NWLockPhysicalRecordSet cannot lock a record that is already locked exclusively by another application. If one or more records, identified in the log table, are already exclusively locked by another application, the attempt to lock the set fails.

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

NWClearPhysicalRecord, **NWClearPhysicalRecordSet**, **NWLogPhysicalRecord**, **NWReleasePhysicalRecord**, **NWReleasePhysicalRecordSet**

NWLogFileLock2

Logs the specified file in the File Log Table and locks the file if the lock flag is set

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWLogFileLock2 (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path,
   nuint8          lockFlags,
   nuint16         timeOut);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWLogFileLock2
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   lockFlags : nuint8;
   timeOut : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle in which the file to be logged resides.

path

(IN) Points to the string containing the path and file name of the file to be logged.

lockFlags

(IN) Specifies the lock flags.

timeOut

(IN) Specifies the length of time the NetWare server attempts to log the specified file before timing out.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x897F	ERR_LOCK_WAITING
0x8982	NO_OPEN_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89FE	TIMEOUT_FAILURE
0x89FF	LOCK_ERROR

Remarks

NWLogFileLock2 logs the specified file for exclusive use by the workstation. If bit 0 of *lockFlags* is set, the server immediately attempts to lock the file.

lockFlags' values are interpreted as follows:

```
0x00   Log file
0x01   Log and lock the file
```

When *lockFlags* is 1, the server attempts to lock the file for the length of time specified by *timeOutLimit*.

path can specify either a file's complete path name or a path relative to the current working directory. For example, if a file's complete path name is SYS:ACCOUNT/DOMEST/TARGET.DAT and the directory handle mapping is SYS:ACCOUNT, *path* could point to either of the following:

```
SYS:ACCOUNT/DOMEST/TARGET.DAT
DOMEST/TARGET.DAT
```

timeOut is specified in units of 1/18 second (0 = no wait).

In DOS and Windows, all access to the network is blocked during any time out period. For this reason, time outs should be kept to an absolute minimum---a value of 18 or less. (Even though DOS is mono-tasking, the application may be running in a DOS box under Windows Enhanced Mode.)

NWLogFileLock2 cannot lock files already logged and exclusively locked by other applications. A file can be locked by a client even if the file does not yet exist. This reserves the file name for use by the client locking it.

The File Log Table contains data locking information used by a NetWare server. The NetWare server tracks this information for each workstation and process. Whenever a file, logical record, or physical record is logged, information identifying the data being logged is entered in the log table. Normally, a set of files or records is logged and then locked as a set. However, a single file or record can also be locked when it is entered in the table.

When using log tables, a task first logs all of the files or records that are needed to complete a transaction. The task then attempts to lock the logged set of files or records. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

NCP Calls

0x2222 03 Log File
0x2222 23 17 Get File Server Information

See Also

NWClearFileLock2, **NWClearFileLockSet**, **NWLockFileLockSet**, **NWReleaseFileLock2**

NWLogLogicalRecord

Logs a logical record in a log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWLogLogicalRecord (
   NWCONN_HANDLE   conn,
   pnstr8          logRecName,
   nuint8          lockFlags,
   nuint16         timeOut);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWLogLogicalRecord
  (conn : NWCONN_HANDLE;
   logRecName : pnstr8;
   lockFlags : nuint8;
   timeOut : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

logRecName

(IN) Points to the name of the logical record being logged (128 characters).

lockFlags

(IN) Specifies the lock flags.

timeOut

(IN) Specifies the length of time the NetWare server attempts to lock the record before timing out.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89FE	TIMEOUT_FAILURE
0x89FF	LOCK_ERROR

Remarks

A logical record is simply a name (string) registered with the NetWare server. The name (as with a semaphore) can then be locked or unlocked by applications and can be used as an inter-application locking mechanism.

NOTE: Locking or unlocking a logical record does not physically lock or unlock those resources associated with the logical record; only the applications using the record know about such an association.

When *lockFlags* is set to option one or three, the NetWare server attempts to lock the logical record for the length of time specified by *timeOut*. *timeOut* is specified in 1/18 second units.

lockFlags' values are the following:

```
0 = Log only
1 = Log and lock exclusive
3 = Log and lock shareable
```

timeOut is specified in units of 1/18 second (0 = no wait).

In DOS and Windows, all access to the network is blocked during any time out period. For this reason, time outs should be kept to an absolute minimum---a value of 18 or less. (Even though DOS is mono-tasking, the application may be running in a DOS box under Windows Enhanced Mode.)

Normally, a set of files or records is logged and then locked as a set. However, a single file or record can also be locked when it is placed in the log table. The release functions, **NWReleaseLogicalRecord** and **NWReleaseLogicalRecordSet**, are used to unlock a lock (or set of locks). The clear functions, **NWClearLogicalRecord** and **NWClearLogicalRecordSet**, are used to unlock and remove a lock (or set of locks) from the log table.

To avoid deadlock, request those resources needing to be locked by making an entry in the File Log Table at the NetWare server. Once the log table is complete, **NWLogLogicalRecord** attempts to lock those records. Locking works only if all records in the table are available. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

NWLogLogicalRecord cannot lock files already logged and exclusively locked by other applications.

NCP Calls

0x2222 09 Log Logical Record

See Also

NWClearLogicalRecord, **NWClearLogicalRecordSet**, **NWLockLogicalRecordSet**, **NWReleaseLogicalRecord**, **NWReleaseLogicalRecordSet**

NWLogPhysicalRecord

Logs a physical record in preparation for a lock
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWLogPhysicalRecord (
   NWFILE_HANDLE   fileHandle,
   nuint32         recStartOffset,
   nuint32         recLength,
   nuint8          lockFlags,
   nuint16         timeOut);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWLogPhysicalRecord
  (fileHandle : NWFILE_HANDLE;
   recStartOffset : nuint32;
   recLength : nuint32;
   lockFlags : nuint8;
   timeOut : nuint16
) : NWCCODE;
```

Parameters

fileHandle

(IN) Specifies the file handle of the file whose record is being logged (must be valid).

recStartOffset

(IN) Specifies the offset into the file where the record being logged begins.

recLength

(IN) Specifies the length, in bytes, of the record to be logged.

lockFlags

(IN) Specifies the lock flags.

timeOut

(IN) Specifies the length of time the NetWare server attempts to lock the record before timing out.

Return Values

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

0x0000	SUCCESSFUL
0x0006	INVALID_HANDLE
0x8988	INVALID_FILE_HANDLE
0x8996	SERVER_OUT_OF_MEMORY
0x89FD	LOCK_COLLISION
0x89FE	TIMEOUT_FAILURE
0x89FF	LOCK_ERROR

Remarks

The NetWare server attempts to log the record for the length of time specified by *timeOutLimit* before returning a time out error. *timeOut* is specified in units of 1/18 second (0 = no wait).

lockFlags' values follow:

```
0 = Log only
1 = Log and lock exclusive
3 = Log and lock shareable
```

timeOut is specified in units of 1/18 second (0 = no wait).

In DOS and Windows, all access to the network is blocked during any timeout period. For this reason, timeouts should be kept to an absolute minimum---a value of 18 or less. (Even though DOS is mono-tasking, the application may be running in a DOS box under Windows Enhanced Mode.)

Normally, a set of files or records is logged and then locked as a set. However, a single file or record can also be locked when it is entered in the log table.

The release functions, **NWReleasePhysicalRecord** and **NWReleasePhysicalRecordSet**, unlock a lock or set of locks. The clear functions, **NWClearPhysicalRecord** and **NWClearPhysicalRecordSet**, unlock and remove a lock or set of locks from the log table.

To avoid deadlock, request those resources needing to be locked by making an entry in the File Log Table at the NetWare server. Once the log table is complete, **NWLogPhysicalRecord** can then lock those records. The locking works only if all records in the table are available. If some of the logged resources cannot be locked, the lock fails and none of the resources are locked.

NWLogPhysicalRecord returns 0x0006 if an invalid file handle is passed to the *fileHandle* parameter.

NCP Calls

0x2222 26 Log Physical Record

See Also

NWClearLogicalRecord, **NWClearLogicalRecordSet**, **NWLockLogicalRecordSet**, **NWReleaseLogicalRecord**, **NWReleaseLogicalRecordSet**

NWOpenSemaphore

Creates and initializes a named semaphore to the indicated value
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWOpenSemaphore (
   NWCONN_HANDLE   conn,
   pnstr8          semName,
   nint16          initSemHandle,
   pnuint32        semHandle,
   pnuint16        semOpenCount);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWOpenSemaphore
  (conn : NWCONN_HANDLE;
   semName : pnstr8;
   initSemHandle : nint16;
   semHandle : pnuint32;
   semOpenCount : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

semName

(IN) Points to the name of the semaphore to be opened.

initSemHandle

(IN) Specifies the number of tasks that can simultaneously access the resources to which the semaphore is tied.

semHandle

(OUT) Points to the NetWare semaphore handle.

semOpenCount

(OUT) Points to the number of stations that currently have this semaphore open (optional; set to NULL if you do not wish this number to be returned).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89FF	LOCK_ERROR

Remarks

Calling **NWOpenSemaphore** increments the *semOpenCount* counter. If the semaphore exists, *initSemHandle* is ignored. The handle returned must be used to access the semaphore. Only the first application to open the semaphore (and thus create the semaphore) can set the initial value in *initSemHandle*.

NWOpenSemaphore is usually called by setting *initSemHandle* to a value other than 0. If *initSemHandle* is set to 0, consider the following items:

Semaphore ownership will not be established until the semaphore is signaled.

The semaphore cannot be used until it is first signaled by an application.

Usually semaphore applications loop from waiting on a semaphore to signaling the semaphore. If *initSemHandle* is 0, the semaphore must be signaled from outside the wait/signal loop.

NWWaitOnSemaphore decrements the semaphore value by 1 if it is greater than 0. If the semaphore value and the *timeOutValue* parameter are both 0, a time out failure (LOCK_ERROR) will be returned.

NWSignalSemaphore increments the semaphore value by 1.

NCP Calls

0x2222 32 Open Semaphore

See Also

NWCloseSemaphore, NWExamineSemaphore, NWSignalSemaphore, NWWaitOnSemaphore

NWReleaseFileLock2

Unlocks the specified file but does not remove it from the log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWReleaseFileLock2 (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          path);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWReleaseFileLock2
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle of the new directory's root directory.

path

(IN) Points to the string containing the name and path of the new directory.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH

Remarks

path can specify either a file's complete path name or a path relative to the current working directory. For example, if a file's complete path name is SYS:ACCOUNT/DOMEST/TARGET.DAT and the directory handle mapping is SYS:ACCOUNT, *path* could be either of the following:

```
SYS:ACCOUNT/DOMEST/TARGET.DAT or
DOMEST/TARGET.DAT
```

2.x servers return INVALID_PATH when a bad directory handle is passed.

NWReleaseFileLock2 is ignored if the requesting workstation does not have locked files.

NCP Calls

0x2222 05 Release File

See Also

NWClearFileLock2, **NWClearFileLockSet**, **NWLogFileLock2**, **NWReleaseFileLockSet**

NWReleaseFileLockSet

Unlocks all files logged in the log table but does not remove them from the table

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWReleaseFileLockSet (
   void);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWReleaseFileLockSet
  : NWCCODE;
```

Return Values

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

0x0000	SUCCESSFUL

Remarks

To avoid deadlock, a workstation must request those files it needs to lock; it does so by making an entry into the File Log Table at the NetWare server. Once the log table is complete, the application can then lock those files. The locking works only if all files in the table are available.

NWReleaseFileLockSet is ignored if the requesting workstation does not have locked files.

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

NWClearFileLock2, **NWClearFileLockSet**, **NWLogFileLock2**, **NWReleaseFileLock2**

NWReleaseLogicalRecord

Unlocks a logical record but does not remove it from the log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWReleaseLogicalRecord (
   NWCONN_HANDLE   conn,
   pnstr8          logRecName);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWReleaseLogicalRecord
  (conn : NWCONN_HANDLE;
   logRecName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle containing the logical record.

logRecName

(IN) Points to the name of the logical record being released (128 characters).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

0x89FF	LOCK_ERROR

Remarks

A logical record is simply a name (a string) registered with the NetWare server. The name (as with a semaphore) can then be locked or unlocked by applications and can be used as an inter-application locking mechanism.

NOTE: Locking or unlocking a logical record does not physically lock or unlock those resources associated with the logical record; only the applications using the record know about such an association.

File Log Table contains data locking information used by a NetWare server. The NetWare server tracks this information for each workstation and workstation task. Whenever a file, logical record, or physical record is logged, information identifying the data being logged is placed in the File Log Table. Normally, a set of files or records is logged and then locked as a set. However, a single file or record can also be locked when it is placed in the table.

NWReleaseLogicalRecord is ignored if the requesting workstation has no records to release.

NCP Calls

0x2222 12 Release Logical Record

See Also

NWClearLogicalRecord, **NWClearLogicalRecordSet**, **NWLockLogicalRecordSet**, **NWReleaseLogicalRecordSet**

NWReleaseLogicalRecordSet

Unlocks all the logical records but does not remove them from the log table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWReleaseLogicalRecordSet (
   void);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWReleaseLogicalRecordSet
  : NWCCODE;
```

Return Values

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

0x0000	SUCCESSFUL

Remarks

A logical record is simply a name (a string) registered with the NetWare server. The name (as with a semaphore) can then be locked or unlocked by applications and can be used as an inter-application locking mechanism.

NOTE: Locking or unlocking a logical record does not physically lock or unlock those resources associated with the logical record; only the applications using the record know about such an association.

To avoid deadlock, a workstation is required to request those files it needs to lock; it does so by making an entry into the File Log Table at the

NetWare server. Once the log table is complete, the application can then lock those files. The locking works only if all files in the table are available.

NWReleaseLogicalRecordSet is ignored if the requesting workstation or process does not have locked logical records.

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

NWClearLogicalRecord, **NWClearLogicalRecordSet**, **NWLockLogicalRecordSet**, **NWLogLogicalRecord**, **NWReleaseLogicalRecord**

NWReleasePhysicalRecord

Unlocks the specified physical record currently locked in the log table of the requesting workstation but does not remove it from the table

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWReleasePhysicalRecord (
   NWFILE_HANDLE   fileHandle,
   nuint32         recStartOffset,
   nuint32         recSize);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWReleasePhysicalRecord
  (fileHandle : NWFILE_HANDLE;
   recStartOffset : nuint32;
   recSize : nuint32
) : NWCCODE;
```

Parameters

fileHandle

(IN) Specifies the file handle associated with the file containing the specified record.

recStartOffset

(IN) Specifies the offset, within the file, where the physical record begins.

recSize

(IN) Specifies the length, in bytes, of the record being released.

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x8988	INVALID_FILE_HANDLE
0x89FF	LOCK_ERROR

Remarks

A physical record lock, as opposed to a logical lock, is the actual lock of a specified record relative to a physical file. When a record is locked, it is also entered into a log table. Records are allowed to be locked only if all records in the log table are available for locking. This is done to avoid deadlock.

NWReleasePhysicalRecord is ignored if the requesting workstation or process does not have locked physical records.

NCP Calls

0x2222 28 Release Physical Record

See Also

NWClearPhysicalRecord, **NWClearPhysicalRecordSet**, **NWLockPhysicalRecordSet**, **NWLogPhysicalRecord**, **NWReleasePhysicalRecordSet**

NWReleasePhysicalRecordSet

Unlocks, but does not remove, all records currently logged as physical records in the requesting workstation's log table

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWReleasePhysicalRecordSet (
   void);
```

Pascal Syntax

```
#include <nwfile.inc>

Function NWReleasePhysicalRecordSet
  : NWCCODE;
```

Return Values

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

0x0000	SUCCESSFUL

Remarks

A physical record lock, as opposed to a logical lock, is the actual lock of a specified record relative to a physical file. When a record is locked, it is also entered into a log table. Records are locked only if all records in the log table are available for locking. This is done to avoid deadlock.

NWReleasePhysicalRecordSet is ignored if the workstation does not have locked physical records.

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

NWClearPhysicalRecord, **NWClearPhysicalRecordSet**, **NWLockPhysicalRecordSet**, **NWLogPhysicalRecord**, **NWReleasePhysicalRecord**

NWScanLogicalLocksByConn

Scans for all logical record locks in a specified connection
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanLogicalLocksByConn (
   NWCONN_HANDLE               conn,
   NWCONN_NUM                  connNum,
   pnint16                     iterHandle,
   CONN_LOGICAL_LOCK N_FAR    *logicalLock,
   CONN_LOGICAL_LOCKS N_FAR   *logicalLocks);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWScanLogicalLocksByConn
  (conn : NWCONN_HANDLE;
   connNum : NWCONN_NUM;
   iterHandle : pnint16;
   Var logicalLock : CONN_LOGICAL_LOCK;
   Var logicalLocks : CONN_LOGICAL_LOCKS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

connNum

(IN) Specifies the connection number of the logged-in object to be scanned.

iterHandle

(IN/OUT) Points to the number of the next record to be scanned.

logicalLock

(OUT) Points to CONN_LOGICAL_LOCK (optional).

logicalLocks

(OUT) Points to CONN_LOGICAL_LOCKS.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89C6	NO_CONSOLE_PRIVILEGES
0x89FD	BAD_STATION_NUMBER
0x88FF	Scan Completed

Remarks

The client must have console operator rights to call **NWScanLogicalLocksByConn**.

iterHandle should be set to 0 initially. Each subsequent call returns the number of the next record to be scanned. *iterHandle* returns -1 upon completion and should not be changed during the scan.

CONN_LOGICAL_LOCKS is a buffer and should be passed to subsequent **NWScanLogicalLocksByConn** calls without modification.

If you pass a non-NULL pointer to *logicalLock*, CONN_LOGICAL_LOCKS passes one record at a time to CONN_LOGICAL_LOCK. If you pass a NULL pointer to *logicalLock*, CONN_LOGICAL_LOCKS is filled but no records are passed to CONN_LOGICAL_LOCK.

0x88FF is returned when the last record has been passed to CONN_LOGICAL_LOCK and **NWScanLogicalLocksByConn** is called subsequently.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 223 Get Logical Records By Connection (2.x)
0x2222 23 239 Get Logical Records By Connection (3.x-4.x)

NWScanLogicalLocksByName

Scans for all record locks in a specified logical name
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanLogicalLocksByName (
   NWCONN_HANDLE          conn,
   pnstr8                 logicalName,
   pnint16                iterHandle,
   LOGICAL_LOCK N_FAR    *logicalLock,
   LOGICAL_LOCKS N_FAR   *logicalLocks);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWScanLogicalLocksByName
  (conn : NWCONN_HANDLE;
   logicalName : pnstr8;
   iterHandle : pnint16;
   Var logicalLock : LOGICAL_LOCK;
   Var logicalLocks : LOGICAL_LOCKS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

logicalName

(IN) Points to the logical lock name to be scanned.

iterHandle

(IN/OUT) Points to the number of the next record to be scanned.

logicalLock

(OUT) Points to LOGICAL_LOCK (optional).

logicalLocks

logicalLocks

(OUT) Points to LOGICAL_LOCKS.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88FF	Scan Completed
0x8996	SERVER_OUT_OF_MEMORY
0x89C6	NO_CONSOLE_PRIVILEGES

Remarks

iterHandle should be set to 0 initially. Each subsequent call returns the number of the next record to be scanned. *iterHandle* returns -1 upon completion and should not be changed during the scan.

If *logicalLock* is a NULL pointer, *logicalLocks* returns the records in groups, instead of one by one.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 224 Get Logical Record Information (2.x)
0x2222 23 240 Get Logical Record Information (3.x-4.x)

NWScanPhysicalLocksByConnFile

Scans for all physical record locks by a specified connection on a specified file

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanPhysicalLocksByConnFile (
   NWCONN_HANDLE                conn,
   NWCONN_NUM                   connNum,
   NWDIR_HANDLE                 dirHandle,
   pnstr8                       path,
   nuint8                       dataStream,
   pnint16                      iterHandle,
   CONN_PHYSICAL_LOCK N_FAR    *lock,
   CONN_PHYSICAL_LOCKS N_FAR  *locks);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWScanPhysicalLocksByConnFile
  (conn : NWCONN_HANDLE;
   connNum : NWCONN_NUM;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   dataStream : nuint8;
   iterHandle : pnint16;
   Var lock : CONN_PHYSICAL_LOCK;
   Var locks : CONN_PHYSICAL_LOCKS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

connNum

(IN) Specifies the connection number of the logged-in object to be

scanned.

dirHandle

(IN) Specifies the directory handle associated with the desired directory path.

path

(IN) Points to a full file path (or a path relative to *dirHandle*) specifying the file to be checked. The last item must be a file name.

dataStream

(IN) Specifies the Macintosh name space (for 3.11 and above only) or set to 0:

0 Resource Fork
1 Data Fork

iterHandle

(IN/OUT) Points to the number of the next record to be scanned (set to 0 initially).

lock

(OUT) Points to the CONN_PHYSICAL_LOCK structure.

locks

(OUT) Points to the CONN_PHYSICAL_LOCKS structure.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88FF	Scan Completed
0x8996	SERVER_OUT_OF_MEMORY
0x89FD	BAD_STATION_NUMBER
0x89FF	FILE_NAME_ERROR, NO_FILES_FOUND_ERROR

Remarks

For 2.x, *path* cannot be longer than 14 bytes including the NULL character.

For 2.x and 3.x, a client must have console operator rights to call **NWScanPhysicalLocksByConnFile** or NO_CONSOLE_PRIVILEGES will be returned.

For 4.x, a client can call **NWScanPhysicalLocksByConnFile** to return information about its connection without needing console operator privileges. To return information about other connection numbers, you must have console rights. A client with console privileges can pass any valid connection number to **NWScanPhysicalLocksByConnFile** and receive information about that connection.

iterHandle returns -1 upon completion and must not be changed during the scan.

If *lock* is a NULL pointer, *locks* returns the records in groups, instead of one by one.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 221 Get Physical Record Locks By Connection And File (2.x)
0x2222 23 237 Get Physical Record Locks By Connection And File (3.x-4.x)
0x2222 23 244 Convert Path To Entry
0x2222 62 Scan First

NWScanPhysicalLocksByFile

Scans for all record locks in a specified physical file
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanPhysicalLocksByFile (
   NWCONN_HANDLE           conn,
   NWDIR_HANDLE            dirHandle,
   pnstr8                  path,
   nuint8                  dataStream,
   pnint16                 iterHandle,
   PHYSICAL_LOCK N_FAR    *lock,
   PHYSICAL_LOCKS N_FAR  *locks);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWScanPhysicalLocksByFile
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   path : pnstr8;
   dataStream : nuint8;
   iterHandle : pnint16;
   Var lock : PHYSICAL_LOCK;
   Var locks : PHYSICAL_LOCKS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle associated with the desired directory path.

path

(IN) Points to a full file path (or a path relative to *dirHandle*) specifying the file to be checked.

dataStream

(IN) Specifies the Macintosh name space (for 3.11 and above only) or set to 0:

0 Resource Fork

1 Data Fork

iterHandle

(IN/OUT) Points to the next record to be scanned; must be set to 0 initially.

lock

(OUT) Points to PHYSICAL_LOCK (optional).

locks

(OUT) Points to PHYSICAL_LOCKS.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88FF	Scan Completed
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89C6	NO_CONSOLE_PRIVILEGES
0x89FD	BAD_STATION_NUMBER
0x89FF	NO_FILES_FOUND_ERROR

Remarks

The client must have console operator rights to call **NWScanPhysicalLocksByFile**.

iterHandle returns -1 upon completion, and should not be changed during the scan.

If *lock* is a NULL pointer, *locks* returns the records in groups, instead of one by one.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 222 Get Physical Record Locks By File (2.x)
0x2222 23 238 Get Physical Record Locks By File (3.x-4.x)
0x2222 23 244 Convert Path to Entry

NWScanSemaphoresByConn

Scans information about the semaphores opened by a specified connection
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanSemaphoresByConn (
   NWCONN_HANDLE             conn,
   NWCONN_NUM                connNum,
   pnint16                   iterHandle,
   CONN_SEMAPHORE NWPTR      semaphore,
   CONN_SEMAPHORES NWPTR   semaphores);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWScanSemaphoresByConn
  (conn : NWCONN_HANDLE;
   connNum : NWCONN_NUM;
   iterHandle : pnint16;
   Var semaphore : CONN_SEMAPHORE;
   Var semaphores : CONN_SEMAPHORES
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

connNum

(IN) Specifies the connection number of the logged-in object to be scanned.

iterHandle

(IN/OUT) Points to the number of the next record to be scanned; should be set to 0 initially.

semaphore

(OUT) Points to CONN_SEMAPHORE (optional).

semaphores

(OUT) Points to CONN_SEMAPHORES.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88FF	Scan Completed
0x8996	SERVER_OUT_OF_MEMORY

Remarks

For 2.x and 3.x, you must have console operator privileges to call **NWScanSemaphoresByConn** or NO_CONSOLE_PRIVILEGES will be returned.

For 4.x, a client can call **NWScanSemaphoresByConn** to return information about its connection without needing console operator privileges. To return information about other connection numbers, you must have console rights. A client with console privileges can pass any valid connection number to **NWScanSemaphoresByConn** and receive information about that connection.

iterHandle returns -1 upon completion, and should not be changed during the scan.

If *semaphore* is a NULL pointer, *semaphores* returns the records in a group, instead of one by one.

NWScanSemaphoresByConn returns SUCCESSFUL even when *connNum* is invalid. Call **NWGetFileServerInformation** to return the *maxConns* supported for the specific file server. Only use *connNum* in the range of zero-*maxConns*.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 225 Get Connection's Semaphores (2.x)
0x2222 23 241 Get Connection's Semaphores (3.x-4.x)

See Also

NWCCGetConnInfo, NWGetObjectConnectionNumbers

NWScanSemaphoresByName

Scans information about a semaphore by name
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWScanSemaphoresByName (
   NWCONN_HANDLE       conn,
   pnstr8              semName,
   pnint16             iterHandle,
   SEMAPHORE N_FAR    *semaphore,
   SEMAPHORES N_FAR   *semaphores);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWScanSemaphoresByName
  (conn : NWCONN_HANDLE;
   semName : pnstr8;
   iterHandle : pnint16;
   Var semaphore : SEMAPHORE;
   Var semaphores : SEMAPHORES
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

semName

(IN) Points to the semaphore name to be scanned.

iterHandle

(IN/OUT) Points to the number of the next record to be scanned; should be set to 0 initially.

semaphore

(OUT) Points to SEMAPHORE (optional).

semaphores

(OUT) Points to SEMAPHORES.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x88FF	Scan Completed
0x8996	SERVER_OUT_OF_MEMORY
0x89C6	NO_CONSOLE_PRIVILEGES

Remarks

The client must have console operator rights to call **NWScanSemaphoresByName**.

iterHandle returns -1 upon completion, and should not be changed during the scan.

If *semaphore* is a NULL pointer, *semaphores* returns the records in groups, instead of one by one.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 226 Get Semaphore Information (2.x)
0x2222 23 242 Get Semaphore Information (3.x-4.x)

NWSignalSemaphore

Increments the semaphore value by one
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWSignalSemaphore (
   NWCONN_HANDLE   conn,
   nuint32         semHandle);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWSignalSemaphore
  (conn : NWCONN_HANDLE;
   semHandle : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

semHandle

(IN) Specifies the semaphore handle of the semaphore to be signaled (obtained by calling **NWOpenSemaphore**).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x89F	LOCK_ERROR

Remarks

If another client is waiting on the semaphore, a successful completion code is returned to the waiting client.

An application must call **NWSignalSemaphore** when it finishes accessing the network resource associated with the semaphore. If processes are waiting to use the semaphore, the first process in the queue is released (signaled).

NCP Calls

0x2222 32 3 Signal Semaphore

See Also

NWCloseSemaphore, **NWExamineSemaphore**, **NWOpenSemaphore**, **NWWaitOnSemaphore**

NWWaitOnSemaphore

Waits on a semaphore for a specified time
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Synchronization

Syntax

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

N_EXTERN_LIBRARY( NWCCODE )NWWaitOnSemaphore (
   NWCONN_HANDLE   conn,
   nuint32         semHandle,
   nuint16         timeOutValue);
```

Pascal Syntax

```
#include <nwsync.h>

Function NWWaitOnSemaphore
  Function NWWaitOnSemaphore
(conn : NWCONN_HANDLE;
   semHandle : nuint32;
   timeOutValue : nuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

semHandle

(IN) Specifies the semaphore handle returned by calling **NWOpenSemaphore**.

timeOutValue

(IN) Specifies the length of time the application will wait for the semaphore.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x89FE	TIMEOUT_FAILURE
0x89FF	LOCK_ERROR

Remarks

NWWaitOnSemaphore decrements the semaphore value counter by 1 if it is greater than 0. If the semaphore value counter and the *timeOutValue* parameter are both 0, a time out failure (LOCK_ERROR) will be returned. If the value is 0 before the time out expires, Successful is returned, and the application can access the associated resource.

If the value is <0, **NWWaitOnSemaphore** queues the application for the time interval specified in *timeOutValue*.

timeOutValue indicates how long the NetWare server should wait if the semaphore value is negative. *timeOutValue* is specified in units of 1/18 second (0 = no wait). It has no default value.

NCP Calls

None

See Also

NWCloseSemaphore, **NWExamineSemaphore**, **NWOpenSemaphore**, **NWSignalSemaphore**

Synchronization: Structures

CONN_LOGICAL_LOCK

Returns a connection's logical locks
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16   taskNumber;
   nuint8    lockStatus;
   nstr8     logicalName[128];
} CONN_LOGICAL_LOCK;
```

Pascal Structure

```
Defined in nwsync.inc

CONN_LOGICAL_LOCK = Record
    taskNumber : nuint16;
    lockStatus : nuint8;
    logicalName : Array[0..127] Of nstr8
  End;
```

Fields

taskNumber

lockStatus

Indicates a bit mask describing how the file is locked:

0x01 Locked
0x02 Open shareable
0x04 Logged
0x08 Open Normal
0x40 TTS holding
0x80 Transaction flag set

logicalName

CONN_LOGICAL_LOCKS

Returns a connection's logical lock list
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16   nextRequest;
   nuint16   numRecords;
   nuint8    records[508];
   nuint16   curOffset;
   nuint16   curRecord;
} CONN_LOGICAL_LOCKS;
```

Pascal Structure

```
Defined in nwsync.inc

  CONN_LOGICAL_LOCKS = Record
    nextRequest : nuint16;
    numRecords : nuint16;
    records : Array[0..507] Of nuint8;
    curOffset : nuint16;
    curRecord : nuint16
  End;
```

Fields

nextRequest

numRecords

records

curOffset

curRecord

CONN_PHYSICAL_LOCK

Returns a connection's physical locks
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16   taskNumber;
   nuint8    lockType;
   nuint32   recordStart;
   nuint32   recordEnd;
} CONN_PHYSICAL_LOCK;
```

Pascal Structure

```
Defined in nwsync.inc

CONN_PHYSICAL_LOCK = Record
    taskNumber : nuint16;
    lockType : nuint8;
    recordStart : nuint32;
    recordEnd : nuint32
  End;
```

Fields

taskNumber

Indicates the number of the task using the file.

lockType

Indicates if the file is locked with the following bits being set:

none Not locked
Bit 0 Locked exclusive
Bit 1 Locked shareable
Bit 2 Logged
Bit 6 Lock held by TTS

recordStart

Indicates the byte offset of where the record begins in the file.

recordEnd

Indicates the byte offset of where the record ends in the file.

CONN_PHYSICAL_LOCKS

Returns a connection's physical lock list
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16              nextRequest;
   nuint16              numRecords;
   CONN_PHYSICAL_LOCK   locks[51];
   nuint16              curRecord;
   nuint8               reserved[22];
} CONN_PHYSICAL_LOCKS;
```

Pascal Structure

```
Defined in nwsync.inc

CONN_PHYSICAL_LOCKS = Record
    nextRequest : nuint16;
    numRecords : nuint16;
    locks : Array[0..50] Of CONN_PHYSICAL_LOCK;
    curRecord : nuint16;
    reserved : Array[0..21] Of nuint8
  End;
```

Fields

nextRequest

Is used internally by **NWScanPhysicalLocksByConnFile**.

numRecords

Indicates the number of valid PHYSICAL_LOCKs.

locks

curRecord

Indicates the current PHYSICAL_LOCK to return in *lock*.

reserved

CONN_SEMAPHORE

Returns semaphore information list
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16   openCount;
   nuint16   semaphoreValue;
   nuint16   taskNumber;
   nstr8     semaphoreName[128];
} CONN_SEMAPHORE;
```

Pascal Structure

```
Defined in nwsync.inc

 CONN_SEMAPHORE = Record
    openCount : nuint16;
    semaphoreValue : nuint16;
    taskNumber : nuint16;
    semaphoreName : Array[0..127] Of nstr8
  End;
```

Fields

openCount

semaphoreValue

taskNumber

semaphoreName

CONN_SEMAPHORES

Returns a connection's semaphore list
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16   nextRequest;
   nuint16   numRecords;
   nuint8    records[508];
   nuint16   curOffset;
   nuint16   curRecord;
} CONN_SEMAPHORES;
```

Pascal Structure

```
Defined in nwsync.inc

  CONN_SEMAPHORES = Record
    nextRequest : nuint16;
    numRecords : nuint16;
    records : Array[0..507] Of nuint8;
    curOffset : nuint16;
    curRecord : nuint16
  End;
```

Fields

nextRequest

numRecords

records

curOffset

curRecord

LOGICAL_LOCK

Defines logical lock information
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   NWCONN_NUM   connNumber;
   nuint16      taskNumber;
   nuint8       lockStatus;
} LOGICAL_LOCK;
```

Pascal Structure

```
Defined in nwsync.inc

  LOGICAL_LOCK = Record
    connNumber : NWCONN_NUM;
    taskNumber : nuint16;
    lockStatus : nuint8
  End;
```

Fields

connNumber

taskNumber

lockStatus

Indicates a bit mask describing how the file is locked:

0x01 Locked
0x02 Open shareable
0x04 Logged
0x08 Open Normal
0x40 TTS holding
0x80 Transaction flag set

LOGICAL_LOCKS

Returns a list of logical locks
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16        useCount;
   nuint16        shareableLockCount;
   nuint8         locked;
   nuint16        nextRequest;
   nuint16        numRecords;
   LOGICAL_LOCK   logicalLock[128];
   nuint16        curRecord;
} LOGICAL_LOCKS;
```

Pascal Structure

```
Defined in nwsync.inc

 LOGICAL_LOCKS = Record
    useCount : nuint16;
    shareableLockCount : nuint16;
    locked : nuint8;
    nextRequest : nuint16;
    numRecords : nuint16;
    logicalLock : Array[0..127] Of LOGICAL_LOCK;
    curRecord : nuint16
  End;
```

Fields

useCount

shareableLockCount

locked

nextRequest

numRecords

logicalLock

curRecord

PHYSICAL_LOCK

Returns physical lock information
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16   loggedCount;
   nuint16   shareableLockCount;
   nuint32   recordStart;
   nuint32   recordEnd;
   nuint16   connNumber;
   nuint16   taskNumber;
   nuint8    lockType;
} PHYSICAL_LOCK;
```

Pascal Structure

```
Defined in nwsync.inc

 PHYSICAL_LOCK = Record
    loggedCount : nuint16;
    shareableLockCount : nuint16;
    recordStart : nuint32;
    recordEnd : nuint32;
    connNumber : nuint16;
    taskNumber : nuint16;
    lockType : nuint8
  End;
```

Fields

loggedCount

Indicates the number of tasks having the record logged.

shareableLockCount

Indicates the number of tasks having the record locked shareable.

recordStart

Indicates the byte offset of where the record begins in the file.

recordEnd

Indicates the logical connection having the record locked exclusively.

connNumber

taskNumber

Indicates the task number within the logical connection having the record locked exclusively.

lockType

Indicates whether the record is locked:

0x00 Not locked

0xFE Locked by a file lock

0xFF Locked by begin share file set

PHYSICAL_LOCKS

Returns a list of physical locks
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16          nextRequest;
   nuint16          numRecords;
   PHYSICAL_LOCK    locks[32];
   nuint16          curRecord;
   nuint8           reserved[8];
} PHYSICAL_LOCKS;
```

Pascal Structure

```
Defined in nwsync.inc

  PHYSICAL_LOCKS = Record
    nextRequest : nuint16;
    numRecords : nuint16;
    locks : Array[0..31] Of PHYSICAL_LOCK;
    curRecord : nuint16;
    reserved : Array[0..7] Of nuint8
  End;
```

Fields

nextRequest

Is internal to **NWScanPhysicalLocksByFile**

numRecords

Indicates the number of valid PHYSICAL_LOCKs

locks

curRecord

Indicates the current PHYSICAL_LOCK.

reserved

SEMAPHORE

Returns semaphore information
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   NWCONN_NUM    connNumber;
   nuint16       taskNumber;
} SEMAPHORE;
```

Pascal Structure

```
Defined in nwsync.inc

  SEMAPHORE = Record
    connNumber : NWCONN_NUM;
    taskNumber : nuint16
  End;
```

Fields

connNumber

taskNumber

SEMAPHORES

Returns a list of semaphores
Service: Synchronization
Defined In: nwsync.h

Structure

```
typedef struct
{
   nuint16     nextRequest;
   nuint16     openCount;
   nuint16     semaphoreValue;
   nuint16     semaphoreCount;
   SEMAPHORE   semaphores[170];
   nuint16     curRecord;
} SEMAPHORES;
```

Pascal Structure

```
SEMAPHORES = Record
   nextRequest : nuint16;
   openCount : nuint16;
   semaphoreValue : nuint16;
   semaphoreCount : nuint16;
   semaphores : Array[0..169] Of SEMAPHORE;
   curRecord : nuint16
 End;
```

Fields

nextRequest

openCount

semaphoreValue

semaphoreCount

semaphores

curRecord

Volume

Volume: Guides

Volume: Task Guide

Reading Volume Information

Managing Disk Space

Additional Links

Volume: Functions

Volume: Structures

Parent Topic:

Volume: Guides

Volume: Concept Guide

Volume Introduction

Volume Basics

Managing Disk Space: Example

Volume Information Functions

Volume Utilization and Restriction Functions

Additional Links

Volume: Functions

Volume: Structures

Parent Topic:

Volume: Guides

Volume: Tasks

Reading Volume Information

NetWare® volume information indicates the amount of space available on a volume. It includes the block size (number of sectors per block) and the following totals:

Total blocks available

Total blocks in use

Total directory entries available

Total directory entries in use

It also indicates whether the volume is removable.

Two functions enable you to read volume information, one by means of volume number and the other by directory handle:

NWGetVolumeInfoWithNumber takes a volume number.

NWGetVolumeInfoWithHandle takes a directory handle.

NWGetVolumeStats returns some additional volume information for NetWare 2.2 volumes (such as whether the volume is caching). The function doesn't apply to 3.11 or above.

Similar information is available at the directory level for 3.11 and above using **NWGetDirSpaceInfo**. In addition to block and directory entry totals, this function returns statistics for purgeable blocks.

Parent Topic:

Volume: Guides

Managing Disk Space

With NetWare® 3.11 and above, you can control the total amount of space available to each object within a volume.

NetWare 3.11 and 4.x servers let you restrict the number of 4 KB blocks available to a specified object. One function sets disk space restrictions and two functions read restrictions:

NWSetObjectVolSpaceLimit sets an object's disk space restriction in blocks. On NetWare 4.x servers, the restriction can range from 0 to 0x08000000. On 3.11 servers, the range is from 0 to 0x40000000.

NWGetObjDiskRestrictions returns the restriction for a specified object.

NWScanVolDiskRestrictions2 can be called iteratively to build a list of objects that are assigned disk space restrictions.

To remove restrictions for a specific object on a volume, call **NWRemoveObjectDiskRestrictions**.

NWGetDiskUtilization returns the number of files, directories, and blocks an object is using on a volume.

Parent Topic:

Volume: Guides

Related Topics:

Managing Disk Space: Example

Volume: Examples

Managing Disk Space: Example

The following code calls **NWGetObjDiskRestrictions** to find the space restriction for a specified user. Command-line parameters supply the volume and object names. **NWParsePath**, **NWGetVolumeNumber**, and **NWGetObjectID** return the parameters needed to call **NWGetObjDiskRestrictions**.

Finding Space Restrictions

```
/* ********************************************************************
 *
 * Name         : Finding space restrictions for a user on a volume
 *
 *
 * Abstract     : Call NWGetObjDiskRestrictions to find space restrictio
 *                for a specified user on a volume (supplied via command
 *                NWParsePath, NWGetVolumeNumber and NWGetObjectID retur
 *                parameters needed to call NWGetObjDiskRestrictions.
 *
 *
 * Inputs       : Usage: SPACE <server> <volumename> <username>
 *
 * Outputs      : restriction information for the user
 *
 * Notes        : This example requires the client to be connected to th
 *                passed in as first argument.
 * ********************************************************************
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <nwnet.h>
#include <nwcalls.h>
#include <nwcaldef.h>
#include <nwdpath.h>
#include <nwclxcon.h>
#ifndef N_PLAT_UNIX
#include <nwdpath.h>
#include <nwvol.h>
#include <nwbindry.h>
#include <nwmisc.h>
#endif
void main(int argc, char *argv[ ])
```

```
{
   NWCONN_HANDLE   connHandle;
   NWCONN_HANDLE   startConnHandle;
     /* buffers for the NWGetObjDiskRestrictions call */
   nuint32         restriction, inUse;
   nuint16         volNumber;
   nuint32         objectID;
   NWCCODE         ccode = 0;

   if (argc < 3)
   {
      printf("\nUsage: SPACE server volname username\n");
      exit(1);
   }
   /* argv[1] = server, argv[2] = vol, argv[3] = object name */
   ccode = NWCallsInit(NULL, NULL);
   if(ccode)
   {
      printf("\nNWCallsInit error %x\n", ccode);
      exit(1);

     /* Get the connection handle for this server */
     ccode = NWCCOpenConnByName((NWCONN_HANDLE)&startConnHandle, argv[1
     NWCC_NAME_FORMAT_NDS, NWCC_OPEN_LICENSED, NULL,
                  (unsigned int *)&connHandle);

     /* Get the volume number for this volume */
     ccode = NWGetVolumeNumber(connHandle, argv[2], &volNumber);
     if (ccode)
     {
        printf("\nNWGetVolumeNumber error %X\n", ccode);
        exit(1);
     }
     printf("\nNWGetVolumeNumber shows %d for %s\n", volNumber, argv[2]

     /* Look up the object id in the bindery.  connHandle is the NetWar
     server connection handle to use, argv[2] specifies the object
     name to search for, OT_USER indicates the bindery type of the obje
     in the search and objectID is the return */

     ccode = NWGetObjectID(connHandle, argv[3], OT_USER, &objectID);
     if (ccode)
     {
        printf("\nNWGetObjectID error %X\n", ccode);
        exit(1);
     }
     /* Get the disk restrictions on a volume for the specified bindery
     object. The number of 4K blocks the user can use will be returned
     in restrictions, the number of 4K blocks the users is currently
     using will be returned in inUse.  */

     ccode = NWGetObjDiskRestrictions(connHandle, volNumber, objectID,
```

```
                                                &restriction, &inUse);
      if (ccode)
      {
         printf("\nNWGetObjDiskRestrictions error %X\n", ccode);
         exit(1);
      }

      printf("Disk restrictions for %s on %s/%s:\n", argv[3], argv[1], a
/*       If the following is true, then the user has no restrictions */

      if (restriction >= 0x40000000L)
         printf("  No disk space restrictions on this user.\n");
      else
      {
         printf("  Limit:  %10d K\n", (restriction * 4));
      }
      printf("  In use: %10d K\n", (inUse * 4));
   }
}
```

Parent Topic:

Volume: Guides

Volume: Concepts

Volume Basics

This section explains the basic concepts related to theNetWare® volumes.

A NetWare volume is the highest level in the NetWare directory structure (the network equivalent of a DOS root directory). Volumes are divided into blocks made up of sectors. Each sector is 512 bytes. The default block size is 4 KB. (The number of blocks per volume depends on the size of the volume.)

A server running versions of NetWare before 3.11 can accommodate up to 32 volumes. A server running NetWare 3.11 or above can accommodate up to 64 volumes.

A NetWare server identifies volumes by name and number. Knowing either value allows you to find the other.

NWGetVolumeNumber uses the volume name to return the volume number.

NWGetVolumeName uses the volume number to find the volume name.

Parent Topic:

Volume: Guides

Volume Information Functions

These functions return information about a volume.

Function	Header	Comment
NWGetVolumeInfoWith Handle	nwvol.h	Returns information for the volume on which the specified directory is found.
NWGetVolumeInfoWith Number	nwvol.h	Returns volume information for the specified volume.
NWGetVolumeName	nwvol.h	Returns the name of the volume associated with the specified volume number.

NWGetVolumeNumber	nwvol.h	Returns the volume number based on the NetWare® server connection ID and volume name.
NWGetVolumeStats	nwvol.h	Returns information about a volume on a 2.2 NetWare server.
NWGetExtendedVolumeInfo	nwvol.h	Returns extended information for the specified volume.

Parent Topic:

Volume: Guides

Volume Introduction

Volume enables you to manage NetWare® volumes. The principle operations performed by these functions include:

Returning information about a specified volume

Accessing space restrictions for a specified Bindery object on a specified volume

Accessing utilization statistics for a specified volume

Although this chapter generally notes the differences between overlapping functions, developers need to be aware of compatibility issues affecting specific functions. To verify a function's compatibility, see Volume: Functions.

For a description of structures and other data definitions that relate to Volume, see Volume: Structures.

Parent Topic:

Volume: Guides

Volume Utilization and Restriction Functions

These functions access space restrictions and utilization statistics for a volume.

Function	Header	Comment
NWGetDiskUtilization	nwvol.h	Returns disk usage for a

		specified bindery object on a volume.
NWGetObjDiskRestrictions	nwvol.h	Returns the restriction on a volume for the specified bindery object.
NWRemoveObjectDiskRestrictions	nwvol.h	Removes all disk restrictions for the specified object on a volume.
NWScanVolDiskRestrictions2	nwvol.h	Returns a list of bindery objects and their disk restrictions on a volume.
NWSetObjectVolSpaceLimit	nwvol.h	Adds a user disk space restriction to a volume.

Parent Topic:

Volume: Guides

Volume: Functions

NWGetDiskUtilization

Allows a client to determine how much physical space the specified object ID is using on the given volume

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWGetDiskUtilization (
   NWCONN_HANDLE   conn,
   nuint32         objID,
   nuint8          volNum,
   pnuint16        usedDirectories,
   pnuint16        usedFiles,
   pnuint16        usedBlocks);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWGetDiskUtilization
  (conn : NWCONN_HANDLE;
   objID : nuint32;
   volNum : nuint8;
   usedDirectories : pnuint16;
   usedFiles : pnuint16;
   usedBlocks : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

objID

(IN) Specifies the object ID.

volNum

(IN) Specifies the volume number.

usedDirectories

(OUT) Points to the number of directories on the volume owned by *objID*.

usedFiles

(OUT) Points to the number of files on the volume owned by *objID*.

usedBlocks

(OUT) Points to the number of physical volume blocks occupied by files owned by *objID*.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x89A1	DIRECTORY_IO_ERROR
0x89F2	NO_OBJECT_READ_PRIVILEGE
0x89FC	NO_SUCH_OBJECT

Remarks

usedBlocks will return incorrect information for disks larger than 268 megabytes. Call **NWGetObjDiskRestrictions** to get the disk space being used by an object.

Clients who are SUPERVISOR equivalent can call **NWGetDiskUtilization** for any object. Clients not having SUPERVISOR rights can call **NWGetDiskUtilization** only for the object used when logging in.

Call either **NWGetObjectID** or **NWMapNameToID** to get the object ID.

NWGetDiskUtilization will not validate *objID*. If *objID* is invalid or does not exist on the server, **NWGetDiskUtilization** will return zero (0) for the disk utilization.

NCP Calls

0x2222 23 14 Get Disk Utilization
0x2222 23 54 Get Object Name

See Also

NWGetObjDiskRestrictions

NWGetExtendedVolumeInfo

Returns extended volume information
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWGetExtendedVolumeInfo (
   NWCONN_HANDLE               conn,
   nuint16                     volNum,
   NWVolExtendedInfo N_FAR  *volInfo);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWGetExtendedVolumeInfo
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   Var volInfo : NWVolExtendedInfo
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number.

volInfo

(OUT) Points to NWVolExtendedInfo, which receives information.

Return Values

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

0x0000	SUCCESSFUL
0x8998	VOLUME_DOES_NOT_EXIST
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x89FB	NO_SUCH_PROPERTY

Remarks

NWGetExtendedVolumeInfo must be called for a licensed connection or NO_SUCH_PROPERTY will be returned.

If **NWGetExtendedVolumeInfo** is called from a 3.x server, NO_SUCH_PROPERTY will be returned.

Possible *volType* values are defined below:

```
0   VINetWare386
1   VINetWare286
2   VINetWare386v30
3   VINetWare386v31
```

Bit definitions for *statusFlag* are shown below:

C Value	Pascal Value	Value Name
0x01	$01	NWSubAllocEnableBit
0x02	$02	NWCompressionEnabledBit
0x04	$04	NWMigrationEnableBit
0x08	$08	NWAuditingEnabledBit
0x10	$10	NWReadOnlyEnableBit

NCP Calls

0x2222 22 51 Get Extended Volume Information

NWGetObjDiskRestrictions

Returns the disk restrictions on a volume for the specified object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWGetObjDiskRestrictions (
   NWCONN_HANDLE   conn,
   nuint8          volNumber,
   nuint32         objectID,
   pnuint32        restriction,
   pnuint32        inUse);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWGetObjDiskRestrictions
  (conn : NWCONN_HANDLE;
   volNumber : nuint8;
   objectID : nuint32;
   restriction : pnuint32;
   inUse : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNumber

(IN) Specifies the volume number for which to return the restrictions.

objectID

(IN) Specifies the object ID.

restriction

(OUT) Points to the buffer containing the number of blocks the object can use.

inUse

(OUT) Points to the buffer containing the number of blocks the object is currently using.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST

Remarks

The restrictions are returned in units of 4K blocks.

NOTE: If the restriction is greater than 0x40000000 on a 3.1 server or 0x80000000 on a 4.x server, the object has no restrictions.

NCP Calls

0x2222 22 41 Get Object Disk Usage And Restrictions

See Also

NWGetExtendedVolumeInfo, **NWSetObjectVolSpaceLimit**

NWGetVolumeInfoWithHandle

Returns the physical information or data of a server's volumes
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWGetVolumeInfoWithHandle (
   NWCONN_HANDLE   conn,
   NWDIR_HANDLE    dirHandle,
   pnstr8          volName,
   pnuint16        totalBlocks,
   pnuint16        sectorsPerBlock,
   pnuint16        availableBlocks,
   pnuint16        totalDirEntries,
   pnuint16        availableDirEntries,
   pnuint16        volIsRemovableFlag);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWGetVolumeInfoWithHandle
  (conn : NWCONN_HANDLE;
   dirHandle : NWDIR_HANDLE;
   volName : pnstr8;
   totalBlocks : pnuint16;
   sectorsPerBlock : pnuint16;
   availableBlocks : pnuint16;
   totalDirEntries : pnuint16;
   availableDirEntries : pnuint16;
   volIsRemovableFlag : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

dirHandle

(IN) Specifies the directory handle pointing to the directory on the volume whose information is to be reported.

volName

(OUT) Points to the volume name (optional 17 character buffer including the terminating NULL).

totalBlocks

(OUT) Points to the total number of blocks on the volume (optional).

sectorsPerBlock

(OUT) Points to the number of sectors per block (optional).

availableBlocks

(OUT) Points to the total number of unused blocks on the volume (optional).

totalDirEntries

(OUT) Points to the total number of physical directory entries (optional).

availableDirEntries

(OUT) Points to the number of unused directory entries (optional).

volIsRemovableFlag

(OUT) Points to a flag indicating whether the volume is removable (optional).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899B	BAD_DIRECTORY_HANDLE
0x899C	INVALID_PATH
0x89FF	HARDWARE_FAILURE

Remarks

NWGetVolumeInfoWithHandle returns a 16-bit number in the *totalBlocks* parameter. If the volume size is greater than a 16-bit number (or 256 megabytes), **NWGetDirSpaceInfo** should be called.

dirHandle is an index number (1 through 255) pointing to a volume,

directory, or subdirectory on the NetWare server. Directory handles are recorded in the Directory Handle Table maintained by the server for each logged-in workstation. When a workstation allocates a directory handle, the NetWare server enters the volume number and directory entry number for the specified directory into the Directory Handle Table. Applications running on the workstation can then refer to a directory using a directory handle, which is actually an index into the Directory Handle Table.

Since all of the output parameters are optional, substitute NULL for unwanted information. However, all parameter positions must be filled.

Volumes use logical sector sizes of 512 bytes. If the physical media uses a different sector size, the server performs appropriate mappings. Volume space is allocated in groups of sectors called blocks.

sectorsPerBlock indicates how many 512-byte sectors are contained in each block of the specified volume.

totalDirEntries indicates how many directory entries were allocated for the specified volume during installation. If this information is meaningless under a given server's implementation, it is 0xFFFF.

volIsRemovableFlag indicates whether a user can physically remove the volume from the NetWare server. It returns one of the following values:

```
0x0000 = not removable/fixed media
non-zero = removable/mountable
```

With NetWare 4.x and SFTIII, the volume sector size can be changed from the 512-byte default. If changed, **NWGetVolumeInfoWithHandle** may return adjusted data meeting DOS requirements. *totalBlocks*, *sectorsPerBlock* and *availableBlocks* may be affected. To see the actual field size, call **NWGetExtendedVolumeInfo**.

NOTE: Block size can be found by calling **NWGetExtendedVolumeInfo** and multiplying *sectorSize* and *sectorPerCluster*.

NCP Calls

0x2222 22 21 Get Volume Info With Handle

See Also

NWGetDirSpaceInfo, **NWGetVolumeInfoWithNumber**

NWGetVolumeInfoWithNumber

Returns information for the specified volume by passing a volume number, allowing a client to check the physical space available on a volume

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWGetVolumeInfoWithNumber (
   NWCONN_HANDLE   conn,
   nuint16         volNum,
   pnstr8          volName,
   pnuint16        totalBlocks,
   pnuint16        sectorsPerBlock,
   pnuint16        availableBlocks,
   pnuint16        totalDirEntries,
   pnuint16        availableDirEntries,
   pnuint16        volIsRemovableFlag);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWGetVolumeInfoWithNumber
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   volName : pnstr8;
   totalBlocks : pnuint16;
   sectorsPerBlock : pnuint16;
   availableBlocks : pnuint16;
   totalDirEntries : pnuint16;
   availableDirEntries : pnuint16;
   volIsRemovableFlag : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

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

volName

(OUT) Points to the volume name (optional 17 character buffer including the terminating NULL).

totalBlocks

(OUT) Points to the total number of blocks on the volume (optional).

sectorsPerBlock

(OUT) Points to the number of sectors per block (optional).

availableBlocks

(OUT) Points to the number of unused blocks on the volume (optional).

totalDirEntries

(OUT) Points to the total number of physical directory entries (optional).

availableDirEntries

(OUT) Points to the number of unused directory entries (optional).

volIsRemovableFlag

(OUT) Points to a flag indicating whether the volume is removable (optional).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST

Remarks

NWGetVolumeInfoWithNumber returns a 16-bit number in the *totalBlocks* parameter. If the volume size is greater than a 16-bit number (or 256 megabytes), **NWGetDirSpaceInfo** should be called.

volNum identifies the volume name on the NetWare server's Volume Table.

Volumes use logical sector sizes of 512 bytes. If the physical media uses a different sector size, the server performs appropriate mappings. Volume

space is allocated in groups of sectors called blocks.

sectorsPerBlock indicates the number of 512-byte sectors contained in each block of the specified volume.

totalDirEntries indicates how many directory entries were allocated for the specified volume during installation. If this information is meaningless under a given server's implementation, it is 0xFFFF.

volIsRemovableFlag indicates whether a user can physically remove the volume from the NetWare server. It returns one of the following values:

0x0000 = not removable/fixed media
non-zero = removable/mountable

Since all of the output parameters are optional, substitute a NULL for unwanted information. However, all parameter positions must be filled.

With NetWare 4.x and SFTIII, the volume sector size can be changed from the 512-byte default. If changed, **NWGetVolumeInfoWithHandle** may return adjusted data that meets DOS requirements. *totalBlocks*, *sectorsPerBlock*, and *availableBlocks* may be affected. To see the actual field size, call **NWGetExtendedVolumeInfo**.

NOTE: Block size can be found by calling **NWGetExtendedVolumeInfo** and multiplying *sectorSize* and *sectorPerCluster*.

NCP Calls

0x2222 18 Get Volume Info With Number

See Also

NWGetDirSpaceInfo, **NWGetVolumeInfoWithHandle**

NWGetVolumeName

Returns the name of the volume associated with the specified volume number and NetWare server

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWGetVolumeName (
   NWCONN_HANDLE   conn,
   nuint16         volNum,
   pnstr8          volName);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWGetVolumeName
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   volName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

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

volName

(OUT) Points to the volume name (17 characters including the terminating NULL).

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST
0x89FF	HARDWARE_FAILURE

Remarks

volNum identifies the volume name on the NetWare server's Volume Table. *volNum* needs to be between 0 and the maximum allowable volumes on the server.

NWGetVolumeName can be called to determine all volume numbers and volume names currently mounted on the specified NetWare server. Start the scan with volume number 0 and scan upwards until

```
volName[0]=="\0"
```

indicating the volume is not mounted.

SUCCESSFUL will be returned for each allowable volume number whether or not that volume exists on the specified server. For example, NetWare 3.x and above supports 64 volumes on each server. Calling **NWGetVolumeName** on each of the 64 volumes will return SUCCESSFUL even though the volume is not mounted.

NCP Calls

0x2222 22 6 Get Volume Name

See Also

NWGetVolumeNumber

NWGetVolumeNumber

Returns the volume number based on the NetWare server connection handle and the volume name

Local Servers: blocking

Remote Servers: blocking

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

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWGetVolumeNumber (
   NWCONN_HANDLE   conn,
   pnstr8          volName,
   pnuint16        volNum);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWGetVolumeNumber
  (conn : NWCONN_HANDLE;
   volName : pnstr8;
   volNum : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volName

(IN) Points to the volume name (17 characters including the terminating NULL).

volNum

(OUT) Points to the volume number (identifies the volume on the NetWare server's Volume Table).

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST

NCP Calls

0x2222 22 5 Get Volume Number

See Also

NWGetVolumeName, NWGetVolumeInfoWithNumber

NWGetVolumeStats

Returns information about a volume
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWGetVolumeStats (
   NWCONN_HANDLE      conn,
   nuint8             volNum,
   VOL_STATS N_FAR  *volInfo);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWGetVolumeStats
  (conn : NWCONN_HANDLE;
   volNum : nuint8;
   Var volInfo : VOL_STATS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

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

volInfo

(OUT) Points to VOL_STATS receiving the volume information.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x899C	INVALID_PATH
0x89FF	HARDWARE_FAILURE

NCP Calls

0x2222 23 233 Get Volume Information

See Also

NWGetVolumeInfoWithHandle, NWGetVolumeInfoWithNumber

NWRemoveObjectDiskRestrictions

Removes any disk restrictions for the specified object, for the specified volume, on the specified server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWRemoveObjectDiskRestrictions (
   NWCONN_HANDLE   conn,
   nuint8          volNum,
   nuint32         objID);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWRemoveObjectDiskRestrictions
  (conn : NWCONN_HANDLE;
   volNum : nuint8;
   objID : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number for which to remove restrictions.

objID

(IN) Specifies the object ID for which to remove restrictions.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	N0_MODIFY_PRIVILEGES
0x8998	VOLUME_DOES_NOT_EXIST
0x89FE	NetWare Error (object has no restrictions)

NCP Calls

0x2222 22 34 Remove User Disk Space Restriction

NWScanVolDiskRestrictions2

Returns a list of the disk restrictions for a volume
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWScanVolDiskRestrictions2 (
   NWCONN_HANDLE               conn,
   nuint8                      volNum,
   pnuint32                    iterHnd,
   NWVOL_RESTRICTIONS N_FAR  *volInfo);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWScanVolDiskRestrictions2
  (conn : NWCONN_HANDLE;
   volNum : nuint8;
   iterhandle : pnuint32;
   Var volInfo : NWVOL_RESTRICTIONS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number for which to return the restrictions.

iterHnd

(OUT) Points to the sequence number to use in the search. Initially must be set to 0.

volInfo

(OUT) Points to NWVOL_RESTRICTIONS.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST

Remarks

NWScanVolDiskRestrictions2 replaces **NWScanVolDiskRestrictions**. The new function uses a larger structure for the volume restrictions that allows up to 16 restrictions per volume.

NOTE: Calling **NWScanVolDiskRestrictions** when you have more than 12 restrictions per volume causes random failures. For this reason, we suggest you call **NWScanVolDiskRestrictions2** exclusively from now on.

The information returned in NWVOL_RESTRICTIONS contains the object restrictions that have been made for the volume. All restrictions are returned in blocks. If the restriction is greater than 0x40000000 on a 3.1 server or 0x80000000 on a 4.x server, the object has no restrictions.

IMPORTANT: **NWScanVolDiskRestrictions2** is called iteratively to retrieve information on all disk space restrictions. The number of entries is returned in *iterHnd*. This value must be added to the previous *iterHnd* to obtain the value for the next iterative call.

NCP Calls

0x2222 22 32 Scan Volume's User Disk Restrictions

NWSetObjectVolSpaceLimit

Sets an object's disk space limit on a volume
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Volume

Syntax

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

N_EXTERN_LIBRARY( NWCCODE ) NWSetObjectVolSpaceLimit (
   NWCONN_HANDLE   conn,
   nuint16         volNum,
   nuint32         objID,
   nuint32         restriction);
```

Pascal Syntax

```
#include <nwvol.inc>

Function NWSetObjectVolSpaceLimit
  (conn : NWCONN_HANDLE;
   volNum : nuint16;
   objID : nuint32;
   restriction : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

(IN) Specifies the volume number for which to set the space limit.

objID

(IN) Specifies the object ID for which to limit the volume space.

restriction

(IN) Specifies the number of blocks to limit the volume space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x898C	N0_MODIFY_PRIVILEGES
0x8996	SERVER_OUT_OF_MEMORY
0x8998	VOLUME_DOES_NOT_EXIST

Remarks

The restrictions are set in units of blocks. The valid space limits range from 0 to 0x08000000.

NOTE: Block size can be found by calling **NWGetExtendedVolumeInfo** and multiplying *sectorSize* and *sectorPerCluster*.

NCP Calls

0x2222 22 33 Add User Disk Space Restriction

Volume: Structures

NWOBJ_REST

Contains an object ID with the restrictions placed on the object for a certain volume (to be used with NWVOL_RESTRICTIONS)

Service: Volume

Defined In: nwvol.h

Structure

```
typedef struct
{
   nuint32   objectID;
   nuint32   restriction;
} NWOBJ_REST;
```

Pascal Structure

```
Defined in nwvol.inc

  NWOBJ_REST = Record
    objectID : nuint32;
    restriction : nuint32
  End;
```

Fields

objectID

Specifies the Directory Services ID for an object.

restriction

Specifies, by the number of blocks, the amount of restriction placed on the object.

NWVolExtendedInfo

Contains extended information for a volume
Service: Volume
Defined In: nwvol.h and nwvol.inc

Structure

```
typedef struct {
   nuint32   volType;
   nuint32   statusFlag;
   nuint32   sectorSize;
   nuint32   sectorsPerCluster;
   nuint32   volSizeInClusters;
   nuint32   freeClusters;
   nuint32   subAllocFreeableClusters;
   nuint32   freeableLimboSectors;
   nuint32   nonfreeableLimboSectors;
   nuint32   availSubAllocSectors;
   nuint32   nonuseableSubAllocSectors;
   nuint32   subAllocClusters;
   nuint32   numDataStreams;
   nuint32   numLimboDataStreams;
   nuint32   oldestDelFileAgeInTicks;
   nuint32   numCompressedDataStreams;
   nuint32   numCompressedLimboDataStreams;
   nuint32   numNoncompressibleDataStreams;
   nuint32   precompressedSectors;
   nuint32   compressedSectors;
   nuint32   numMigratedDataStreams;
   nuint32   migratedSectors;
   nuint32   clustersUsedByFAT;
   nuint32   clustersUsedByDirs;
   nuint32   clustersUsedByExtDirs;
   nuint32   totalDirEntries;
   nuint32   unusedDirEntries;
   nuint32   totalExtDirExtants;
   nuint32   unusedExtDirExtants;
   nuint32   extAttrsDefined;
   nuint32   extAttrExtantsUsed;
   nuint32   DirectoryServicesObjectID;
   nuint32   volLastModifiedDateAndTime;
} NWVolExtendedInfo;
```

Pascal Structure

```
NWVolExtendedInfo = Record
    volType : nuint32;
    statusFlag : nuint32;
```

```
  sectorSize : nuint32;
  sectorsPerCluster : nuint32;
  volSizeInClusters : nuint32;
  freeClusters : nuint32;
  subAllocFreeableClusters : nuint32;
  freeableLimboSectors : nuint32;
  nonfreeableLimboSectors : nuint32;
  availSubAllocSectors : nuint32;
  nonuseableSubAllocSectors : nuint32;
  subAllocClusters : nuint32;
  numDataStreams : nuint32;
  numLimboDataStreams : nuint32;
  oldestDelFileAgeInTicks : nuint32;
  numCompressedDataStreams : nuint32;
  numCompressedLimboDataStreams : nuint32;
  numNoncompressibleDataStreams : nuint32;
  precompressedSectors : nuint32;
  compressedSectors : nuint32;
  numMigratedDataStreams : nuint32;
  migratedSectors : nuint32;
  clustersUsedByFAT : nuint32;
  clustersUsedByDirs : nuint32;
  clustersUsedByExtDirs : nuint32;
  totalDirEntries : nuint32;
  unusedDirEntries : nuint32;
  totalExtDirExtants : nuint32;
  unusedExtDirExtants : nuint32;
  extAttrsDefined : nuint32;
  extAttrExtantsUsed : nuint32;
  DirectoryServicesObjectID : nuint32;
  volLastModifiedDateAndTime : nuint32
End;
```

Fields

volType

Specifies different volumes that may be supported in the future.

statusFlag

Specifies the options currently available in this volume

C Value	Pascal Value	Value Name
0x01	$01	NWSubAllocEnableBit
0x02	$02	NWCompressionEnabledBit
0x04	$04	NWMigrationEnableBit
0x08	$08	NWAuditingEnabledBit
0x10	$10	NWReadOnlyEnableBit

sectorSize

Specifies the sector size in bytes.

sectorsPerCluster

Specifies the number of sectors per cluster.

volSizeInClusters

Specifies the size, in clusters, of the volume.

freeClusters

Specifies the number of clusters currently free for allocation. This does not include space currently available from deleted (limbo) files, nor space that could be reclaimed from the suballocation file system.

subAllocFreeableClusters

Specifies the space that could be reclaimed from the suballocation file system.

freeableLimboSectors

Specifies the disk space, in clusters, that could be freed from deleted files.

nonfreeableLimboSectors

Specifies the disk space, in clusters, currently in deleted files, not aged enough to be classified as *FreeableLimboClusters*. These will be migrated to the status of *FreeableLimboCluster* after time.

availSubAllocSectors

Specifies the space available to the suballocation file system, but not freeable to return as clusters.

nonuseableSubAllocSectors

Specifies the disk space wasted by the suballocation file system. These clusters cannot be allocated by the suballocation system or used as regular clusters.

subAllocClusters

Specifies the disk space being used by the suballocation file system.

numDataStreams

Specifies the number of data streams for real files with data allocated to them.

numLimboDataStreams

Specifies the number of data streams for deleted files with data allocated to them.

oldestDelFileAgeInTicks

Specifies the current age of the oldest file in ticks.

numCompressedDataStreams

Specifies the number of data streams for compressed real files.

numCompressedLimboDataStreams

Specifies the count of data streams for compressed deleted files.

numNoncompressibleDataStreams

Specifies the data streams found not compressable (real and deleted).

precompressedSectors

Specifies the disk space allocated to all files before they were compressed (includes "hole" space).

compressedSectors

Specifies the disk space used by all compressed files.

numMigratedDataStreams

Specifies the number of migrated data streams.

migratedSectors

Specifies the migrated disk space (in sectors).

clustersUsedByFAT

Specifies the disk space in clusters used by the FAT table.

clustersUsedByDirs

Specifies the disk space in clusters used by directories.

clustersUsedByExtDirs

Specifies the disk space in clusters used by the extended directory space.

totalDirEntries

Specifies the total number of directories available on the volume.

unusedDirEntries

Specifies the total directory entries unused on volume.

totalExtDirExtants

Specifies the amount of extended directory space extants (128 bytes each) available on volume.

unusedExtDirExtants

Specifies the amount of extended directory space extants (128 bytes each) unused on volume.

extAttrsDefined

Specifies the number of extended attributes defined on volume.

extAttrExtantsUsed

Specifies the number of extended directory extants used by the extended attributes.

DirectoryServicesObjectID

Specifies the Directory Services ID for volume.

volLastModifiedDateAndTime

Specifies the last time any file or subdirectory within the volume was

modified (tracked by the OS).

Remarks

The *volType* parameter can have the following values:

0 VINetWare386
1 VINetWare286
2 VINetWare386v30
3 VINetWare386v31

NWVOL_RESTRICTIONS

Returns a list of objects with space restrictions on a volume
Service: Volume
Defined In: nwvol.h

Structure

```
typedef struct
{
   nuint8   numberOfEntries;
   struct
   {
      nuint32   objectID;
      nuint32   restriction;
   } resInfo[16];
} NWVOL_RESTRICTIONS;
```

Pascal Structure

```
Defined in nwvol.inc

  NWVOL_RESTRICTIONS = Record
    numberOfEntries : nuint8;
    resInfo : Array[0..15] Of RES_INFO;
  End;

  RES_INFO = Record
    objectID : nuint32;
    restriction : nuint32;
  End;
```

Fields

numberOfEntries

Specifies the number of objects in the list (0-16 objects).

objectID

Specifies the ID of the NDS object.

restriction

Specifies the size, in blocks, of the restriction placed on an object.

VOL_STATS

Contains volume statistics
Service: Volume
Defined In: nwvol.h

Structure

```
typedef struct
{
   nint32   systemElapsedTime;
   nuint8    volumeNumber;
   nuint8    logicalDriveNumber;
   nuint16   sectorsPerBlock;
   nuint16   startingBlock;
   nuint16   totalBlocks;
   nuint16   availableBlocks;
   nuint16   totalDirectorySlots;
   nuint16   availableDirectorySlots;
   nuint16   maxDirectorySlotsUsed;
   nuint8    isHashing;
   nuint8    isCaching;
   nuint8    isRemovable;
   nuint8    isMounted;
   nstr8     volumeName[16];
} VOL_STATS;
```

Pascal Structure

```
Defined in nwvol.inc

VOL_STATS = Record
    systemElapsedTime : nint32;
    volumeNumber : nuint8;
    logicalDriveNumber : nuint8;
    sectorsPerBlock : nuint16;
    startingBlock : nuint16;
    totalBlocks : nuint16;
    availableBlocks : nuint16;
    totalDirectorySlots : nuint16;
    availableDirectorySlots : nuint16;
    maxDirectorySlotsUsed : nuint16;
    isHashing : nuint8;
    isCaching : nuint8;
    isRemovable : nuint8;
    isMounted : nuint8;
    volumeName : Array[0..15] Of nstr8
  End;
```

Fields

systemElapsedTime

Specifies how long the server has been up. This value is returned in ticks (units of approximately 1/18 second) and is used to determine the amount of time elapsing between consecutive calls. After reaching a value of 0xFFFFFFFF, the value wraps back to zero.

volumeNumber

Specifies the number of a volume in a volume table on a server. SYS volume is always zero.

logicalDriveNumber

Specifies the logical drive number of the drive on which the volume exists.

sectorsPerBlock

Specifies the number of 512-byte sectors contained in each block of the specified volume. For NetWare 2.x, this is configurable from 1 to 16. NWU (NetWare for Unix) does not support this field and returns a zero.

startingBlock

Specifies the number of the first block of the volume.

totalBlocks

Specifies the number of blocks in the specified volume. NWU (NetWare for Unix) returns the total amount of disk space on the volume's host file system. All volumes mounted from the same file system will return the same value.

availableBlocks

Specifies the number of unused blocks in the specified volume. NWU (NetWare for Unix) returns the total amount of disk space on the host file system. All volumes mounted from the same file system will return the same value.

totalDirectorySlots

Specifies the number of directory slots allocated for the specified volume. NWU (NetWare for Unix) returns the number of files that can be created to track NetWare file and trustee information.

availableDirectorySlots

Specifies the number of directories that can be created, based on the differences between the total allowable number of directories and the number of directories already created. NWU (NetWare for Unix) returns the number of directories that can be created.

maxDirectorySlotsUsed

Specifies the greatest number of directory slots ever used at one time on the volume. NWU (NetWare for Unix) does not support this field and returns a zero.

isHashing

Specifies whether the volume is hashing in server memory (0=not hashing). Only NetWare 2.x servers return a valid value.

isCaching

Specifies whether the volume is caching in server memory (0=volume not caching). Only NetWare 2.x servers returns a valid value.

isRemovable

Specifies if a user can physically remove the volume from the server (0=cannot be removed). Only NetWare 3.x and 4.x servers return a valid value.

isMounted

Specifies whether the volume is physically mounted in the server (0=volume is not mounted). Only NetWare 3.x and 4.x servers return a valid value.

volumeName

Specifies the name given to the volume (1 to 16 characters long). It cannot contain asterisks (*), question marks (?), colons (:), slashes (/), or backslashes (\). If the name is less than 16 characters, the remaining characters must be null. NWU (NetWare for Unix) returns the NetWare name for the volume.

Volume Management

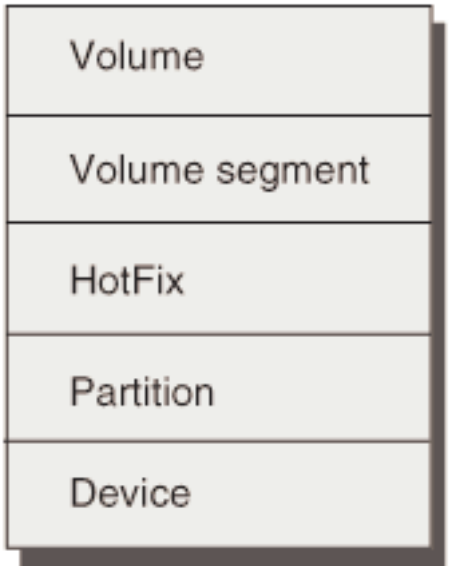
Volume Management: Guides

Volume Management: Concept Guide

Storage Layers

The NetWare® storage system generally consists of these layers:

Figure 10. NetWare Storage Systems Layers



Device Layer

Partition Layer

Hot Fix Layer

Volume Segment Layer

Volume Layer

Parent Topic:

Volume Management: Guides

Volume Segments

A volume segment has two important aspects: it is a piece of a volume and a piece of a mirror object. The volume aspect relates to "What disks does this volume span?", while the mirror object aspect relates to "What volumes are on this disk?" Volumes and segments can be represented physically or logically, as explained below.

Physical Representation

Logical Representation

Segment Types

Parent Topic:

Volume Management: Guides

Volume Management: Concepts

Compatibility

These functions were written to NetWare® 4.x and NetWare 3.12. These functions cannot be used with NetWare 3.11 or earlier, or with Personal NetWare™.

Directory Services and volume control capabilities such as suballocation, compression, and migration do not exist in NetWare 3.12. As a result, **NWVL_SetOrGetVolumeControlInfo** always returns an error on NetWare 3.12. The *volumeControlFlags* and *dsObjectID* parameters of **NWVL_CreateANewVolume** are NetWare 4.x-specific, and are ignored on NetWare 3.12.

When creating new volumes on NetWare 4.x, you can get defaults for block size and the volume control flag by calling **NWVL_GetDefaultBlockSizeAndControlFlags**. The suballocation volume control flag is typically returned as "on"; there seems to be little advantage to setting it "off." Compression and migration flags have performance vs. disk space trade-offs; you may want to leave those issues to the user.

After creating a volume object on NetWare 4.x, you should get the Directory Services ID by calling DSAPI. Because DSAPI depends on DS, and DS cannot be properly installed without volume SYS being installed first, an interesting "chicken-and-egg" scenario can result. To solve this, call **NWVL_CreateANewVolume** with *dsObjectID* set to 0; then after DS is installed, call **NWVL_SetOrGetVolumeControlInfo** to set the correct ID.

Parent Topic:

Volume Management: Guides

Device Layer

The device, described in Media Manager: Guides, corresponds to the drive, magazine, or player. The media corresponds to the actual storage medium in the device.

Parent Topic:

Storage Layers

Related Topics:

Partition Layer

Hot Fix Layer

Volume Segment Layer

Volume Layer

Hot Fix Layer

The Hot Fix™ layer is a fault-tolerant layer designed to handle media errors. Up to 2% of the partition is reserved for redirection (Hot Fix); the remainder contains the data. If a media error occurs while writing a data block (4 KB block size), the entire block is redirected to the Hot Fix area, and the original block in the data area will remain unused. A mirror object is a group of one or more Hot Fix objects. Each Hot Fix object contains identically redundant data. Reads can occur from any of them, and writes are directed to all of them.

Parent Topic:

Storage Layers

Related Topics:

Volume Segment Layer

Volume Layer

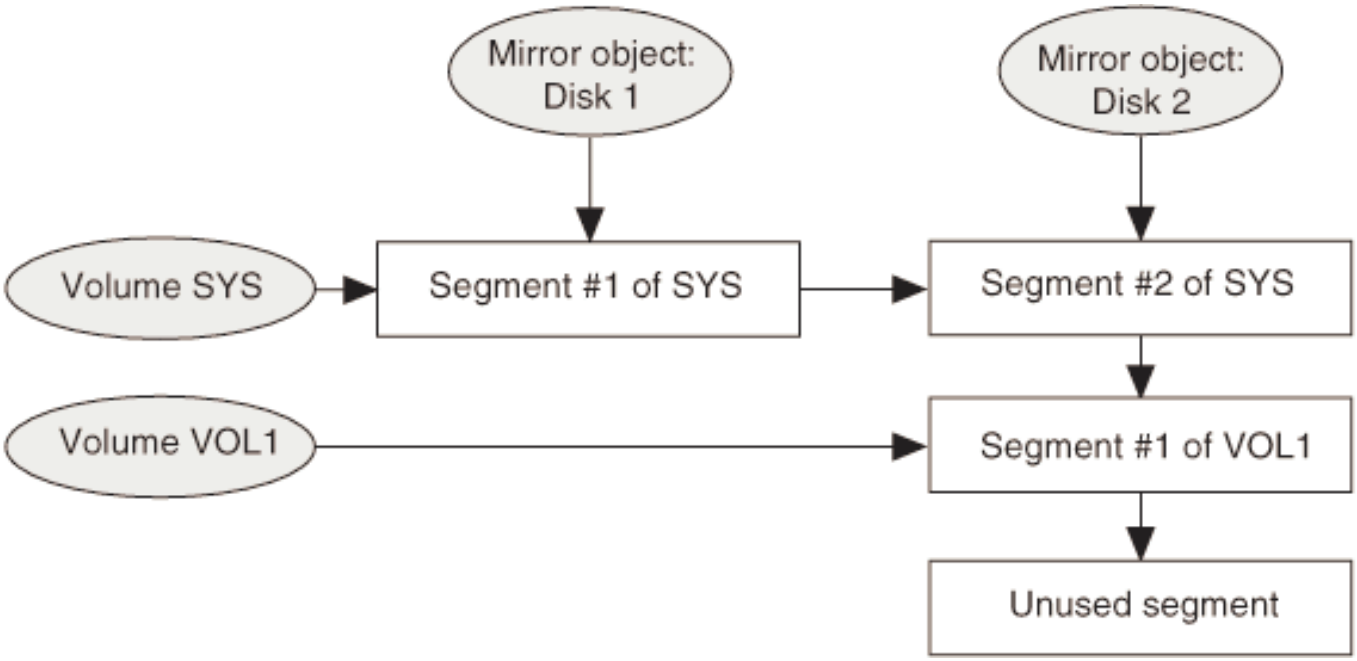
Device Layer

Partition Layer

Logical Representation

These two aspects can be thought of as the horizontal and vertical dimensions of an imaginary grid, as shown in the following figure. The horizontal dimension represents the volume aspect. An array of volume names forms the headers to the list, with links to segments across one or more disks. The volume list head is obtained by calling **NWVL_GetVolumeFirstSegment**, then the segment list can be traversed by calling **NWVL_GetNextSegment** with the appropriate traversal flag.

Figure 11. Volume Segments, Logical Representation



The vertical dimension represents the mirror object. An array of mirror objects forms the linked list headers, which point to a list of segments within each mirror object. The partition list head is obtained by calling **NWVL_GetMirrorObjectFirstSegment**, then the segment list can be traversed by calling **NWVL_GetNextSegment** with the appropriate traversal flag.

Parent Topic:

Volume Segments

Related Topics:

Segment Types

Physical Representation

Mounting Volumes

The NetWare® OS mounts a volume by scanning the disks for the list of complete volumes. (The API and the NetWare OS use similar code.) Then the NetWare OS reads the FAT and directory structure into memory, making the internal volume structure (directories, files, and so on) accessible to the user.

The NetWare OS itself (NetWare 3.12 and NetWare 4.x) does not mount individual segments, only complete volumes. Therefore, if a segment is invalid it is not mountable. Also, the (current) NetWare OS does not mount more than one volume with the same name, even though (according to the API) the volumes may be complete, valid, and distinct. (The API implementation internally uses more than just the name to determine volume uniqueness.)

Parent Topic:

Volume Management: Guides

Partition Layer

A partition follows the IBM partition table convention, with a maximum of four physical partitions per drive. INSTALL.NLM currently creates only one NetWare® partition per drive, although an application can create more, independent of INSTALL.NLM.

Parent Topic:

Storage Layers

Related Topics:

Hot Fix Layer

Volume Segment Layer

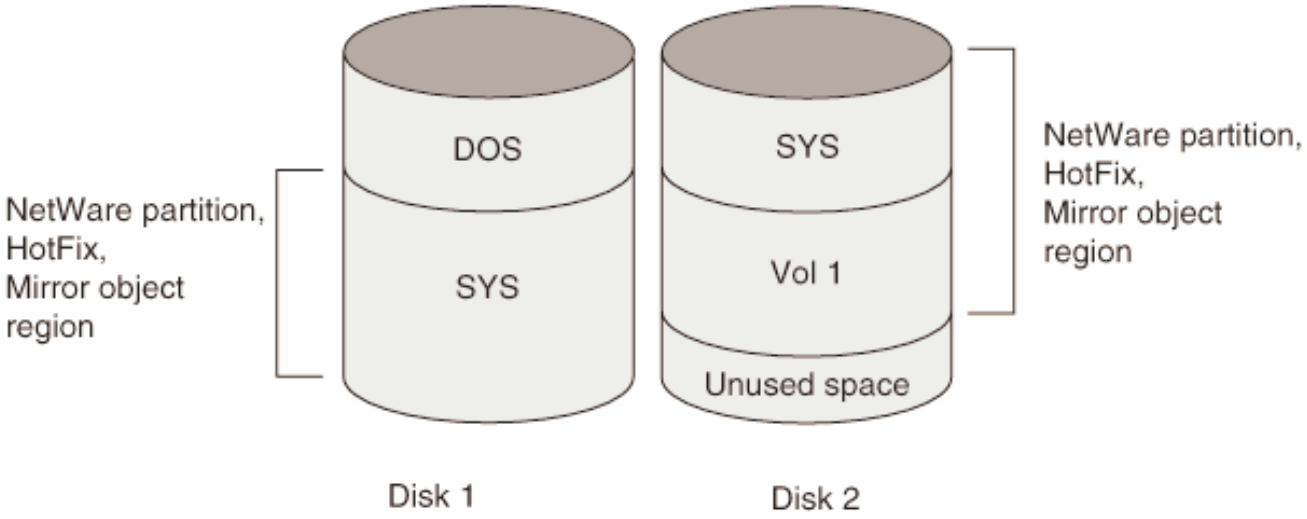
Volume Layer

Device Layer

Physical Representation

Most users prefer to think of the physical representation of volume segments, because typically they create volumes with only one segment, and one segment per disk. Therefore, for most typical users, a volume equals a disk. A physical representation of volume segments is shown below.

Figure 12. Volume Segments, Physical Representation



Parent Topic:

Volume Segments

Related Topics:

Logical Representation

Segment Types

Segment Status Codes

Error Code	Description
NWVL_SEGSTAT_READFAIL	Excessive read errors occurred.
NWVL_SEGSTAT_BUSY	Could not lock the partition.
NWVL_SEGSTAT_READERRORS	Read errors occurred.
NWVL_SEGSTAT_UNUSABLE	Volume definitions

	contradictory.
NWVL_SEGSTAT_INCONSISTENT	Some volume definitions don't make sense.
NWVL_SEGSTAT_DUPLICATE_VOL	Complete, but alternative volume.
NWVL_SEGSTAT_DUPLICATE_SEG	Duplicate segments for an existing volume.
NWVL_SEGSTAT_INCOMPLETE_VOL	Volume fragment; pieces missing.
NWVL_SEGSTAT_INCONSISTENT_VOL	Volume fragment; pieces don't fit.

Parent Topic:

Volume Management: Guides

Segment Types

Segments are classed as valid, invalid, and free. Valid segments form part of complete and consistent volumes. Invalid segments are those whose volume definition is incomplete, corrupted, or extraneous. Unused mirror object space is classed as a free segment.

The list of segments associated with each volume contains only valid volume segments. The segment list for each mirror object contains all segments, including the free and invalid ones. When you write code using the volume list, remember that the list may not be comprehensive (even though in normal, error-free conditions it probably is).

Parent Topic:

Volume Segments

Related Topics:

Physical Representation

Logical Representation

Translation Routines

For future compatibility, allowance has been made for 64-bit offsets (NWVL_QUAD) and Unicode* volume names. Translation routines are provided to translate between NWVL_QUAD and unsigned 32-bit values. Additional translation routines are provided for translating between local and Unicode volume names. For DAI backward-compatibility, translation routines to go back and forth between mirror object numbers and logical

partition numbers.

Parent Topic:

Volume Management: Guides

Volume Creation Error Codes

Error Code	Description
NWVLERR_ACCESS_NOT_ALLOWED	Shared access does not allow the attempted operation.
NWVLERR_ALLOC	Could not allocate memory.
NWVLERR_API_VERSION	Desired API version level is not supported.
NWVLERR_BUSY	**NWVL_Start** was not called first, or function already in use.
NWVLERR_CANT_GET_OS_VER	Could not get OS server version.
NWVLERR_CANT_GET_VOLINFO	Could not get OS volume information.
NWVLERR_CANT_REGISTER_EVENT	Unable to register for an OS event.
NWVLERR_CANT_RESET_COMPR	Compression is on, cannot turn it off.
NWVLERR_CANT_RESET_SUBALLOC	Suballocation is on, cannot turn it off.
NWVLERR_CREATE_DIR	Was not able to create the volume directory table.
NWVLERR_CREATE_FAT	Was not able to create the volume FAT table.
NWVLERR_DEFREADAFTERWRITE	Unable to read-after-write verify the vol def table.
NWVLERR_DEFREALWRITE	Unable to write the vol def table.
NWVLERR_DEFWRITE	Unable to write the vol def table.
NWVLERR_DEFWRITEVERIFY	Unable to pre-read-before write the vol def table.
NWVLERR_DIR_ALLOC	A free directory block could not be found (the volume may be full).
NWVLERR_DIR_INCONSISTENT	The directory table is

	inconsistent.
NWVLERR_DIR_READ	A directory block could not be read.
NWVLERR_DIR_REDIR	An attempt to redirect a directory block write failed.
NWVLERR_DIR_WRITE	A directory block could not be written.
NWVLERR_DUPLICATE_SEGMENT	The new segment matches an existing one.
NWVLERR_EXPAND_VOLTABLES	Cannot expand OS volume FAT and directory tables.
NWVLERR_FAIL_DISMOUNT	Cannot dismount the volume.
NWVLERR_FAIL_MOUNT	Cannot mount the volume.
NWVLERR_FAT_ALLOC	A free FAT block could not be found (the volume may be full).
NWVLERR_FAT_INCONSISTENT	The fat length read is the incorrect size, or the primary and mirror fat don't match.
NWVLERR_FAT_READ	Unable to read the volume FAT tables.
NWVLERR_FAT_WRITE	Unable to write the volume FAT/directory tables.
NWVLERR_GET_INFO	An attempt to get partition information failed.
NWVLERR_INTERNAL	An unexpected internal error occurred.
NWVLERR_INVALID_BLOCKSHIFT	Invalid block shift factor, must be 3-7.
NWVLERR_INVALID_CALLER	**NWVL_Start** was not called, or caller ID was not found.
NWVLERR_INVALID_CHARS	Name contains one or more invalid characters. Valid characters include A-Z, 0-9, @, #, %, &, (, and).
NWVLERR_INVALID_OFFSET	The offset specified cannot be used.
NWVLERR_INVALID_LOGICAL	Invalid logical partition number.
NWVLERR_INVALID_PARAM	A parameter passed to a function was invalid.
NWVLERR_INVALID_PARTITION	An invalid partition object number was specified.

NWVLERR_INVALID_SEGMENT	An invalid segment handle was specified.
NWVLERR_INVALID_VOLUME	An invalid volume handle was specified.
NWVLERR_IS_FREE	The specified segment cannot be a free segment.
NWVLERR_LONG_NAME	The volume name cannot be more than 15 characters.
NWVLERR_MULTIPLE_UNDERSCORE	The volume name cannot contain multiple adjacent underscores.
NWVLERR_NAME_NOT_NEW	This volume name is already being used.
NWVLERR_NOT_FREE	The specified segment must be a free segment.
NWVLERR_NOT_NETWARE_VOL	The volume specified is not a NetWare® volume.
NWVLERR_NOT_ON_THIS_OS	Call not supported on this OS.
NWVLERR_NO_MORE_SEGMENTS	No more segments found.
NWVLERR_NO_MORE_VOLUMES	No more volumes found.
NWVLERR_NO_PARTITIONS	No NetWare partitions were found.
NWVLERR_NO_SEGMENTS	No volume segments were found.
NWVLERR_OVERLAP	New volume to create crosses over other segment boundaries.
NWVLERR_PART_BUSY	Could not lock the partition.
NWVLERR_PERIOD_IN_NAME	The volume name cannot contain a period.
NWVLERR_RESERVED_NAME	The volume name is a reserved name.
NWVLERR_SECTORS_BELOW_MIN	The number of sectors specified is below the minimum.
NWVLERR_SECTORS_NOT_NORMALIZED	Number of sectors not a multiple of the block size.
NWVLERR_SHORT_NAME	The volume name must be at least two characters long.
NWVLERR_TOO_MANY_SEGMENTS	No additional segments can be added on this partition.
NWVLERR_TOO_MANY_VOLSEG	No additional segments can be added to this volume.

NWVLERR_UNDERSCORE_AT_BEGOREND	The volume name cannot contain an underscore at the beginning or end.
NWVLERR_VOLUME_BUSY	Part of the volume was locked.
NWVLERR_VOLUME_MOUNTED	The volume was mounted and should not have been.
NWVLERR_VOLUME_NOT_MOUNTED	The volume was not mounted and should have been.
NWVLERR_VOL_BUSY	Cannot lock the volume.

Parent Topic:

Volume Management: Guides

Volume Layer

NetWare® volumes consist of groups of one or more volume segments. File I/O accesses occur at the volume layer.

Parent Topic:

Storage Layers

Related Topics:

Device Layer

Partition Layer

Hot Fix Layer

Volume Segment Layer

Volume Management Introduction

> **IMPORTANT:** This document describes software that is subject to change. It is the intention of Novell® to keep it as current as possible, but the user must assume any inherent risk in developing or maintaining code based on the information in this document. Novell might discontinue or decline to support any of the software features described herein.

The Volume Management API is provided in an NLM™ library that runs on NetWare® 3.12 and NetWare 4.x. These functions perform tasks such as the following:

Creating NetWare volumes

Deleting NetWare volumes

Modifying NetWare volumes

Support to applications that install, back up, and recover NetWare volumes

IMPORTANT: To use this API, you also need the VOLLIB.NLM software and the VOLLIB.H header file. To create an application, you also need a NetWare SDK (with the NTYPES.H include file) and the Media Manager SDK.

Parent Topic:

Volume Management: Guides

Volume Segment Layer

A volume segment, the fundamental unit of volume disk storage, is like a subdivision of a logical partition.

Because mirroring is a file system function handled at a lower level than the volume segment, you can assume that a volume segment will be mirrored if it is created on a mirrored mirror object (that is, with more than one Hot Fix object per mirror object). A mirror object may be divided into a maximum of eight volume segments. A volume may include a total of up to 32 segments from any of the mirror objects.

IMPORTANT: Eight and 32 are the current static limits. However, because this may change in the future, you should call the **NWVL_GetSupportedLimits** and **NWVL_GetMirrorObjectSupportedLimits** functions to get the actual values, instead of hard-coding these limits into your application.

Parent Topic:

Storage Layers

Related Topics:

Volume Layer

Device Layer

Partition Layer

Hot Fix Layer

Warning Codes

Warning Code	Description
NWVLWARN_VOLS_NEED_UPDATE	Some volumes definitions found were inconsistent.

Parent Topic:

Volume Management: Guides

Volume Management: Functions

NWVL_ChangeSegmentName

Renames a volume segment
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int   NWVL_ChangeSegmentName (
   nuint32   caller,
   nuint32   segmentHandle,
   pnwchar   newVolName );
```

Parameters

caller

(IN) Calling application

segmentHandle

(IN) Handle of volume segment being renamed

newVolName

(IN) Pointer to new Unicode name for segment

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

The new segment name cannot correspond to the name of any volume currently mounted on the server, or the function fails.

The new segment name need not be unique, but cannot result in a segment that is identical to another segment in both name and ordinal volume position. For example, you can use this function sequentially to rename the segments in a volume, thus creating duplicate new names after the first segment is renamed. However, the ordinal volume positions remain different for each of the renamed segments.

NWVL_ChangeVolumeName

Rename a dismounted volume
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_ChangeVolumeName (
   nuint32   caller,
   nuint32   volumeHandle,
   pnwchar   newVolName );
```

Parameters

caller

(IN) Calling application

volumeHandle

(IN) Handle of volume being renamed

newVolName

(IN) Pointer to new name for volume

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not rename the volume in the Directory. On a DS server, make sure this step is not forgotten; see DSAPI for doing this.

NWVL_ConvertLogicalPartitionToMirrorObject

Converts the NetWare Disk Application Interface (DAI) logical partition number to the NetWare Media manager mirror object ID

Classification: 3.12, 4.x

SMP Aware: No

Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_ConvertLogicalPartitionToMirrorObject (
   nuint32    caller,
   nuint32    logicalPartNumber,
   pnuint32   mirrorObjectID );
```

Parameters

caller

(IN) Calling application

logicalPartNumber

(IN) DAI logical partition number

mirrorObjectID

(OUT) Media manager number for the mirror object

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_ConvertMirrorObjectToLogicalPartition

Converts the NetWare Media manager mirror object ID to the NetWare Disk Application Interface (DAI) logical partition number

Classification: 3.12, 4.x

SMP Aware: No

Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_ConvertMirrorObjectToLogicalPartition (
   nuint32    caller,
   nuint32    mirrorObjectID,
   pnuint32   logicalPartNumber );
```

Parameters

caller

Calling application

mirrorObjectID

Media manager number for the mirror object

logicalPartNumber

DAI logical partition number

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_CreateANewVolume

Creates a new volume
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_CreateANewVolume (
   nuint32     caller,
   pnwchar     volumeName,
   nuint32     blockSize,
   nuint32     volControlFlags,
   nuint32     dsObjectID,
   nuint32     mirrorObjectID,
   NWVL_QUAD   startingSector,
   NWVL_QUAD   sectorsInSegment,
   pnuint32    returnVolHandle );
```

Parameters

caller

(IN) Calling application

volumeName

(IN) Points to name of volume being created. **Volume names must be at least three characters long and follow standard volume naming rules.**

blockSize

One of the following values:

NWVL_4K_BLOCK3
NWVL_8K_BLOCK4
NWVL_16K_BLOCK5
NWVL_32K_BLOCK6
NWVL_64K_BLOCK7

volControlFlags

(IN) Specifies one of the following values:

NWVL_SUB_ALLOCATION_ENABLED_BIT0x02

NWVL_FILE_COMPRESSION_ENABLED_BIT0x04 File compression enabled on this volume.

NWVL_DATA_MIGRATION_ENABLED_BIT0x08 Data migration is allowed on this volume.

dsObjectID

(IN) Directory Services object ID

mirrorObjectID

(IN) Mirror object number

startingSector

(IN) Offset (zero-relative) to first sector of mirror object

sectorsInSegment

(IN) Number of desired sectors in the new segment

returnVolHandle

(OUT) Handle for newly created volume

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

The available space is rounded down to the nearest whole-block number of sectors. This function does not create the volume in the Directory, but will stamp the identifier on the volume once it is created. On a DS server, make sure this step is not forgotten; see DSAPI for doing this. On a 3.x server, the *volumeControlFlags* and *dsObjectID* parameters are ignored.

NWVL_DeleteAVolume

Removes a volume and its segments
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_DeleteAVolume (
   nuint32   caller,
   nuint32   volumeHandle );
```

Parameters

caller

(IN) Calling application

volumeHandle

(IN) Handle for volume being enlarged

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

This function deletes all segments of the volume. The volume must be dismounted.

NWVL_DeleteSegment

Deletes a segment, converting it to free space
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_DeleteSegment (
   nuint32   caller,
   nuint32   segmentHandle);
```

Parameters

caller

(IN) Calling application

segmentHandle

(IN) Handle of segmented being deleted

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

This function is typically needed when separating mirror groups.

NWVL_DismountVolume

Dismounts a volume, through the NetWare OS
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_DismountVolume(
   nuint32   caller,
   pnwchar   volumeName);
```

Parameters

caller

(IN) Calling application

volumeName

(IN/OUT) Unicode name of volume to dismount

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_End

Removes the in-memory volume list and allow a different caller to access the volume list

Classification: 3.12, 4.x

SMP Aware: No

Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_End(
   nuint32   caller);
```

Parameters

caller

(IN) Calling application

Return Values

0 if successful; nonzero error code if unsuccessful

NWVL_EnlargeVolume

Adds space to a volume
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_EnlargeVolume(
   nuint32     caller,
   nuint32     volumeHandle,
   nuint32     mirrorObjectID,
   NWVL_QUAD   startingSector,
   NWVL_QUAD   sectorsInSegment);
```

Parameters

caller

(IN) Calling application

volumeHandle

(IN) Handle for volume being enlarged

mirrorObjectID

(IN) Mirror object number

startingSector

(IN) Offset (zero-relative) to first sector

sectorsInSegment

(IN) Number of available sectors per segment

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

The available space is rounded down to the nearest block.

IMPORTANT: The volume may be either mounted or dismounted. However, if there is more than one volume with the same name, the volume must be dismounted.

NWVL_GetDefaultBlockSizeAndControlFlags

Gets the recommended block size and volume optimization flags for a proposed volume

Classification: 3.12, 4.x

SMP Aware: No

Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_GetDefaultBlockSizeAndControlFlags (
   nuint32     caller,
   NWVL_QUAD   numberOfSectors,
   pnuint32     blockSize,
   pnuint32     controlFlags);
```

Parameters

caller

(IN) Calling application

numberOfSectors

(IN) Number of sectors in volume

blockSize

(OUT) Recommended block size for volume

controlFlags

(OUT) Volume optimization flags for compression, suballocation, or migration.

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_GetMirrorObjectFirstSegment

Returns the first segment of the volume definition table, given the mirror object number

Classification: 3.12, 4.x

SMP Aware: No

Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_GetMirrorObjectFirstSegment (
   nuint32    caller,
   nuint32    mirrorObjectID,
   pnuint32   firstSegmentHandle);
```

Parameters

caller

(IN) Calling application

mirrorObjectID

(IN) Media manager number for the mirror object

firstSegHandle

(IN) Mirror object first segment handle

Return Values

0 if successful; nonzero error code if unsuccessful

NWVL_GetMirrorObjectSupportedLimits

Return a partition's parameter limits for applications
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_GetMirrorObjectSupportedLimits (
   nuint32    caller,
   nuint32    mirrorObjectID,
   pnuint32   maxSegsPerPart,
   pnuint32   minSectorsPerSeg,
   pnuint32   sectorSize);
```

Parameters

caller

(IN) Calling application

mirrorObjectID

(IN) Media manager number for the mirror object

maxSegsPerPart

(OUT) Maximum number of segments allowed per logical partition

minSectorsPerSeg

(OUT) Minimum number of sectors allowed per segment

sectorSize

(OUT) Bytes per sector

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

The parameter limits are needed by applications for checking and displaying information to be passed to the NWVL functions for a particular partition. **This function does not automatically initiate a volume scan.**

NWVL_GetNextSegment

Gets the next volume segment handle
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_GetNextSegment (
   nuint32    caller,
   nuint32    flag,
   pnuint32   nextSegment);
```

Parameters

caller

(IN) Calling application

flag

(IN) Specifies either `NWVL_TRAVERSE_PARTITION_LIST` (get the next segment in the partition) or `NWVL_TRAVERSE_VOLUME_LIST` (get the next segment in the volume).

nextSegment

(IN/OUT) On input, specifies the handle returned from **NWVL_GetVolumeFirstSegment** or **NWVL_GetMirrorObjectFirstSegment**. On output, receives the next volume segment handle

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

The previous volume segment handle is passed in, and the next handle is returned. To get information on the segment, call **NWVL_GetSegmentInfo**.

NWVL_GetSegmentInfo

Returns segment, disk, and volume information, given the segment handle
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_GetSegmentInfo (
   nuint32            caller,
   nuint32            segmentHandle,
   NWVL_SegmentInfo   *segInfo);
```

Parameters

caller

(IN) Calling application

segmentHandle

(IN) Handle to segment being queried

**segInfo*

(OUT) Information from NWVL_SegmentInfo structure (see description below)

caller

(IN) Calling application

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

This function uses the NWVL_SegmentInfo structure:

```
Typedef struct {
     /* General segment information */
   nuint32     type;
     /* NWVL_FREE_SEGMENT, NWVL_VALID_SEGMENT,
      NWVL_INVALID_SEGMENT */
   nuint32   status;/* NWVL_SEGSTAT_READFAIL, etc. */
     /* Segment information associated with the partition. */
   NWVL_QUAD   startingDiskSector;
   NWVL_QUAD   sectorsInSegment;
```

```
nuint32     mirrorObjectID;
  /* Segment information associated with the volume. */
nwchar      volumeName[NWVL_VOLNAME_BUFFER_SIZE];
  /* Unicode, undefined if free */
nuint32     blockSize;  /* undefined if free; NWVL_4K_BLOCK,
                        etc. */
nuint32     numberOfSegments;
  /* 1 - maximum sectors per volume; undefined if free */
nuint32     mySegmentPosition;
  /* 0 through numberOfSegments - 1; undefined if free */
nuint32     volumeFlags;   /* undefined if free */
NWVL_QUAD   blocksInVolume;       /* undefined if free */
nuint32     volumeHandle;  /* undefined if free or invalid
                           segment */
nuint32     sectorSize;   /* bytes per sector in segment */
nuint32     flags;         /* currently unused */
nuint32     reserved[4];    /* undefined */
} NWVL_SegmentInfo;
```

NWVL_GetSupportedLimits

Returns a volume's parameter limits for applications
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_GetSupportedLimits(
   nuint32    caller,
   pnuint32   maxVolNameSize,
   pnuint32   quadBits,
   pnuint32   maxSegsPerVol);
```

Parameters

caller

(IN) Calling application

maxVolNameSizeMax

(OUT) Size of volume name in Unicode characters, including null character

quadBits

(OUT) Bits in a quad

maxSegsPerVol

(OUT) Maximum number of segments allowed in a volume

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

The parameter limits are needed by applications for checking and displaying information to be passed to the NWVL functions.

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_GetVolumeFirstSegment

Gets the first segment in a volume
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_GetVolumeFirstSegment(
   nuint32    caller,
   nuint32    volumeHandle,
   pnuint32   firstSegHandle);
```

Parameters

caller

(IN) Calling application

volumeHandle

(IN) Volume handle

firstSegHandle

(OUT) Volume's first segment handle

Return Values

0 if successful; nonzero error code if unsuccessful

NWVL_IsVolumeMounted

Asks the NetWare OS whether a specified volume is mounted
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_IsVolumeMounted (
   nuint32    caller,
   pnwchar    volumeName,
   pnuint32   mountedFlag);
```

Parameters

caller

(IN) Calling application

volumeName

(IN) Unicode name of volume to query

mountedFlag

(OUT) Returns TRUE in value if mounted

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_LocalToUnicode

Converts a string from the local code page to Unicode* characters
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_LocalToUnicode (
   nuint32   caller,
   pnchar    localChars,
   nuint32   maxUniChars,
   pnwchar   uniChars);
```

Parameters

caller

(IN) Calling application

localChars

(IN) Local code page characters

maxUniChars

(IN) Maximum number of Unicode characters

uniChars

(OUT) String converted to Unicode

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_MountVolume

Mounts a volume, through the NetWare OS
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_MountVolume(
   nuint32   caller,
   pnwchar   volumeName);
```

Parameters

caller

(IN) Calling application

volumeName

(IN/OUT) Unicode name of volume to mount

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_Nuint32ToQuad

Converts a nuint32 value to NWVL_QUAD
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_Nuint32ToQuad (
   nuint32      caller,
   nuint32      value,
   NWVL_QUAD   *quadValue);
```

Parameters

caller

(IN) Calling application

value

(IN) nuint32 value to convert

**quadValue*

(OUT) Converted NWVL_QUAD value

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_QuadToNuint32

Converts a NWVL_QUAD value to nuint32
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_QuadToNuint32 (
   nuint32      caller,
   NWVL_QUAD    quadValue,
   nuint32     *value);
```

Parameters

caller

(IN) Calling application

quadValue

(IN) NWVL_QUAD value to convert

**value*

(OUT) Converted nuint32 value

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan. Sub-allocation units are valid on this volume.

NWVL_SetOrGetVolumeControlInfo

Sets or queries the flags for optimizing volume management
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_SetOrGetVolumeControlInfo(
   nuint32    caller,
   nuint32    volumeHandle,
   nuint32    action,
   pnuint32   controlFlags,
   pnuint32   dsID);
```

Parameters

caller

(IN) Calling application

volumeHandle

(IN) Handle of volume:action 0 = Get the flags' values; 1 = Set the flags' values

controlFlags

(IN/OUT) Set or get volume optimization flags for compression, suballocation, or migration. Passed or returned depending on action; can be NULL.

dsID

(IN/OUT) Set or get the volume's Directory Services ID. Passed or returned depending on the action; can be NULL.

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

The volume may be mounted or dismounted when **NWVL_SetOrGetVolumeControlInfo** is called. However, if more than one volume has the same name, the volume must be dismounted.

IMPORTANT: Once compression and suballocation are turned on, they cannot be turned off.

NWVL_Start

Initializes the volume library functions
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_Start(
   nuint32     identity,
   nuint32     majorVersion,
   nuint32     minorVersion,
   nuint32     lockType,
   pnuint32   *returnedCaller);
```

Parameters

identity

(IN) NLM handle

majorVersion

(IN) `NWVL_MAJOR`, the API major version number

minorVersion

(IN) `NWVL_MINOR`, the API major version number

lockType

(IN) Type of lock to use: `NWVL_SHARED` or `NWVL_EXCLUSIVE`

returnedCaller

(OUT) Handle to use in subsequent VOLLIB calls

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

NWVL_Start Allows exclusive read-write access for one client at a time, or shareable read-only access for multiple clients.

NWVL_UnicodeToLocal

Converts a string from Unicode to the local code page
Classification: 3.12, 4.x
SMP Aware: No
Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_UnicodeToLocal (
   nuint32   caller,
   pnwchar   uniChars,
   nuint32   maxLocalChars,
   pnchar    localChars);
```

Parameters

caller

(IN) Calling application

uniChars

(IN) String converted to Unicode

maxUniChars

(IN) Maximum number of Unicode characters

localChars

(OUT) Local code page characters

Return Values

0 if successful; nonzero error code if unsuccessful

Remarks

IMPORTANT: This function does not automatically initiate a volume scan.

NWVL_ValidateVolumeName

Returns whether a volume name is valid (acceptable to **NWVL_CreateANewVolume**) before creating the volume

Classification: 3.12, 4.x

SMP Aware: No

Service: Volume Management

Syntax

```
#include <vollib.h>

int NWVL_ValidateVolumeName (
   nuint32   caller,
   pnwchar   volName);
```

Parameters

caller

(IN) Calling application

volName

(IN) Unicode name of volume

Return Values

0 if volume name is valid; nonzero error code if volume name is invalid